

MAPEO Y LOCALIZACIÓN BASADA EN ODOMETRÍA USANDO
IMÁGENES COMO LAND MARKS

EDUARDO BORRERO CRUZ

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍAS
PROGRAMA DE INGENIERÍA ELECTRONICA
BOGOTÁ, D. C.
2016

MAPEO Y LOCALIZACIÓN BASADA EN ODOMETRÍA USANDO
IMÁGENES COMO LAND MARKS

EDUARDO BORRERO CRUZ

Trabajo de grado para optar por el título de Ingeniero Electrónico

Director

Andrés Camilo Jiménez Alvarez

MsC. Ingeniería Electrónica

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍAS
PROGRAMA DE INGENIERÍA ELECTRONICA
BOGOTÁ, D. C.
2016

Nota de aceptación

Firma del presidente del jurado

Firma del jurado

Firma del Jurado

BOGOTÁ, D. C. 2016

Las directivas de la Fundación Universitaria Los Libertadores, los jurados calificadores y el cuerpo docente no son responsables por los criterios e ideas expuestas en el presente documento. Estos corresponden únicamente a los autores

DEDICATORIA

A MI FAMILIA

AGRADECIMIENTOS

Al Ingeniero Andrés Jiménez, por el apoyo, confianza, interés y acompañamiento mostrados durante el proceso del desarrollo del presente documento, desde el inicio hasta su culminación, mediante el aporte constante de su conocimiento y experiencia al proyecto.

Próximo Ingeniero Aeronáutico Mario Solórzano, por brindar su ayuda y conocimientos en el desarrollo del proyecto.

Próximo Ingeniero Electrónico Rolando Moreno, por la generosidad con la que brindó su apoyo.

Coordinación de laboratorios, salas y talleres de la Fundación Universitaria Los Libertadores, por la colaboración prestada en la disposición de espacios para el desarrollo del proyecto.

CONTENIDO

	Pág.
LISTA DE TABLAS	9
LISTA DE IMÁGENES	10
LISTA DE DIAGRAMAS	12
LISTA DE ANEXOS	13
OBJETIVOS	14
OBJETIVO GENERAL:	14
OBJETIVOS ESPECIFICOS:	14
PLANTEAMIENTO DEL PROBLEMA	15
JUSTIFICACIÓN	16
ESTADO DEL ARTE	17
1. AGENTE ROBOTICO	20
1.1 Modelamiento cinemático	21
1.2 Restricciones cinemáticas.	26
1.2.1 Primera restricción	26
1.2.2 Segunda restricción	26
1.3.3 Tercera restricción	26
1.3 Modelamiento Dinámico	28
1.3.1 Velocidades de las ruedas.	29
1.3.2 Energías Cinéticas	33
2. Sensores, Actuadores y diseño del agente robótico	38
1.1 Raspberry pi 2 Model b	39
1.2 Sensor HC-SR04	41
1.3 Sensor MPU6050	42
1.4 Sensor HMC5983	43
1.5 Motores Dynamixel Ax-12a	43
1.6 Atmega16	46

1.7	Baterías	48
1.7.1	Batería Li-Po	48
1.7.2	Power Bank	48
1.8	Protocolo de comunicación I2c	49
1.9	Construcción del Robot móvil	51
3.	Procesamiento de imágenes	54
3.1	Odometría	54
3.1.1	Calibración Cámara	55
3.2	Land-marks:	58
3.3	Identificación de los Land-Marks	59
3.3.1	Modelo de colores HSV	59
3.3.2	Técnicas de segmentación	61
3.3.2.1	Binarización de imágenes	61
3.3.2.2	Erosión de imágenes	62
3.3.2.3	Dilatación de imágenes	62
3.3.2.4	Cambio de tamaño de una imagen.	63
3.3.3	Detección del Land-Mark	64
4.	Estimación de Movimiento	71
5.	Mapas	77
	CONCLUSIONES	84
	RECOMENDACIONES	85
	REFERENCIAS	86
	ANEXOS	88

LISTA DE TABLAS

Tabla 1 Especificaciones técnicas de la Raspberry pi 2 model B	40
Tabla 2 Promedio de errores entre los Land-Marks	69
Tabla 3 Comparación error de la Calibración	70
Tabla 4 Valores asignados Ruedas Traseras	72
Tabla 5 Velocidad del robot móvil prueba 1	73
Tabla 6 Velocidad Rueda 1 = 0x52c	73
Tabla 7 Velocidad Rueda 2 = 0x130	73
Tabla 8 Velocidad del Robot Móvil prueba 2	74
Tabla 9 Velocidad Rueda 1 = 0x4ff	74
Tabla 10 Velocidad Rueda 2 = 0x103	74
Tabla 11 Velocidad Robot Movil prueba 3	75
Tabla 12 Velocidad Rueda 1 = 0x496	75
Tabla 13 Velocidad Rueda 2 = 0x9a	75
Tabla 14 Velocidad Robot Móvil Prueba 4	76
Tabla 15 Velocidad Rueda 1 = 0x44b	76
Tabla 16 Velocidad Rueda2 = 4f	76
Tabla 17 Velocidades determinadas	77

LISTA DE IMÁGENES

	Pág.
Imagen 1 Robot Diferencial	20
Imagen 2 Esquema del robot	22
Imagen 3 Masas de las Ruedas y el chasis	29
Imagen 4 Partes Raspberry pi model B	40
Imagen 5 Angulo de medida	41
Imagen 6 : Imagen a diafonía , Imagen b Reflexión	41
Imagen 7 Orientación Sensor MPU650	42
Imagen 8 Circuito Half Duplex UART	44
Imagen 9 Conexión motores con Raspberry pi	44
Imagen 10 Paquete de instrucciones dynamixel ax-12A	45
Imagen 11 Distribución de pines del microcontrolador ATmega16	47
Imagen 12 Batería Li-Po	48
Imagen 13 Batería Power Bank	49
Imagen 14 Estados del Protocolo I2c	50
Imagen 15 Circuito motores Dynamixel e integrado 74ls241	51
Imagen 16 PCB #2	52
Imagen 17 Medidas Robot Móvil	53
Imagen 18 Giro de Rueda	55
Imagen 19 Problemas de la Odometría	55
Imagen 20 Patrón de calibración para la cámara	56
Imagen 21 Detectar esquinas del patrón	57
Imagen 22 Resultado de la Calibración	58
Imagen 23 Land-Marks	58
Imagen 24 Cono Colores HSV.	59
Imagen 25 Ejemplo OPENCV	60
Imagen 26 Tabla calibración del color HSV	60
Imagen 27 Color Utilizado para el Land-mark y la mascara	61
Imagen 28 Binarizacion de una imagen	62
Imagen 29 Erosión	62
Imagen 30 Dilatación	63
Imagen 31 Interpolación	63
Imagen 32 Land-Marks	64
Imagen 33 Rectángulo del contorno del Land-Mark	65
Imagen 34 Land-Mark girado	67
Imagen 35 Proceso Cropped	68
Imagen 36 Comparación de errores de los Land-Marks	69
Imagen 37 Mapa Topológico	78

Imagen 38 ubicación del robot móvil	78
Imagen 39 Recorrido con la mismas velocidad angular en cada rueda	79
Imagen 40 Recorrido del robot con diferentes velocidades angulares	80
Imagen 41 Recorrido del robot, trayectoria predefinida	80
Imagen 42 Escenario para las pruebas del robot	81
Imagen 43 Primera Prueba de Mapeo	81
Imagen 44 Resultado primera Prueba	82
Imagen 45 Segunda prueba de Mapeo	82
Imagen 46 Resultado Segunda prueba	83
Imagen 47 Visualización escritorio Raspberry pi	89
Imagen 48 Raspberry pi Configuraciones	89
Imagen 49 Selección en el programa del uC Atmega16	116
Imagen 50 Selecccion de Programador	117
Imagen 51 Cargar el Programa a el ATmega16	118

LISTA DE DIAGRAMAS

Diagrama 1 Componentes del agente robótico	39
Diagrama 2 Conexión I2c	51
Diagrama 3 Etapas de la odometría	54
Diagrama 4 Algoritmo detectar el Land-Mark	64

LISTA DE ANEXOS

Anexo A Instalación y configuración Raspberry pi:	88
Anexo B Código Sensor Ultrasonido en c	93
Anexo C Librería sensor MPU6050	97
Anexo D liberaría Sensor HCM5983	104
Anexo E Programación de ATmega16 en Arduino IDE.	108
Anexo F Primer ejemplo OPENCV	121
Anexo G Código Calibración de la Cámara	122
Anexo H Librería motores Dynamixel	124
Anexo I Sensores ultrasonido raspberry pi	143
Anexo J Codigo procesamiento de imagenes.	145
Anexo K Algoritmo evasión de obstáculos	151
Anexo L CÓDIGO COMPLETO	159

OBJETIVOS

OBJETIVO GENERAL:

Diseñar un algoritmo de mapeo y localización basada en odometría usando imágenes como land-marks, implementado en un robot móvil de tipo diferencial que sea capaz de percibir y modelar el entorno, para la localización del agente robótico de conocer su posición y orientación el entorno se debe conocerse a priori.

OBJETIVOS ESPECIFICOS:

1. Realizar el modelamiento dinámico y cinemático del robot móvil.
2. Conocer los métodos de segmentación de imágenes, para poder identificar los diferentes Land-Marks y utilizar el método de identificación de imágenes más eficiente.
3. Realizar una plataforma grafica para observar los resultados del tipo de mapa generado por el robot móvil.

PLANTEAMIENTO DEL PROBLEMA

La implementación de un robot móvil, que genere mapas y se localice tiene una gran cantidad de aplicaciones en las que se puede utilizar, por lo anterior es necesario establecer las herramientas que se vayan a utilizar para que su construcción sea económica y los requerimientos de computo sean mínimos.

La investigación sobre la localización en entornos desconocidos y que simultáneamente vaya generando un mapa, ha dejado como resultado un amplio número de avances esencialmente en nuevos algoritmos, diseños de robots móviles y tipos de sensores, para que su rendimiento sea más eficiente de acciones combinadas de localización, exploración y visión artificial para la detección de objetos o marcas.

Por lo tanto ¿Es posible mediante un robot móvil construir un mapa de un entorno mientras simultáneamente usa dicho mapa para calcular su posición y poder localizarse?

JUSTIFICACIÓN

En la actualidad existen plataformas móviles enfocadas al mapeo y localización que utilizan altos recursos computacionales, los cuales son capaces de generar múltiples procesos al mismo tiempo; lo que lleva a que su costo de construcción sea muy elevado por qué se necesitan equipos informáticos de alto nivel.

Debido a esta razón se considera que es necesario implementar nuevas técnicas o algoritmos, los cuales integren múltiples sensores de bajo costo y que sea eficiente desde el punto de vista computacional, también por los distintos procesos que debe realizar al mismo tiempo el agente robótico, se decidió trabajar solo con una cámara, ya que el beneficio de trabajar con una sola cámara es tratar la información en 2-d y no de forma tridimensional.

Plantear un proyecto de este estilo que pueda reunir varias ramas de la ingeniería electrónica como lo es procesamiento de imágenes, circuitos digitales y robótica todo bajo un fundamento matemático fomenta el estudio de la robótica móvil en general en la Fundación Universitaria Los Libertadores, ya que esta área de investigación se encuentra en un proceso de crecimiento muy rápido y en Colombia hay pocos grupos de investigación sobre este tema.

El estudio de la robótica móvil enfatiza en varias ramas como al industria o en la vida doméstica y para esto es muy importante que él pueda ubicarse en su entorno de trabajo, por esta razón surge la línea de investigación llamada SLAM (Simultaneous Localization and Mapping), que es una técnica usada por los robots para poder localizarse y realizar el mapa del entorno en el que se está, dicha línea de investigación se está estudiando para poder generar algoritmos óptimos y de bajo costo para poder resolver este tipo de problemas.

La oportunidad de desarrollar un proyecto de grado que promueva y origine desarrollo e investigación en Colombia, es de gran inspiración y motivación hace que la realización del mismo se desarrolle con gusto y que se pueda determinar énfasis de trabajo para un futuro.

ESTADO DEL ARTE

De acuerdo con lo estipulado por Joaquín Viñals Pons en su trabajo de grado *“Localización y generación de mapas del entorno (SLAM) de un robot por medio de una Kinect”* [1] el nacimiento del problema del SLAM ocurre durante la Conferencia sobre robótica y Automática del IEEE en la ciudad de San Francisco, durante el año de 1986 como resultado de esta conferencia determinan que el mapeo y la localización es un problema fundamental de la robótica, por lo que comienzan a introducir temas fundamentales como lo son los métodos probabilísticos y la inteligencia artificial.

Gracias a estas técnicas de exploración la nasa como una de las entidades en trabajar en este tipo de temas envía robots móviles a marte para la exploración y mapeo de este planeta envía al Mars Pathfinder para tomar datos sobre la atmosfera de marte, el segundo lanzamiento fue el curiosity para poder tomar imágenes del entorno y explorar desde esto se puede ver como el SLAM es fundamental en todas las ramas en la que se trabaje la robótica móvil.

Por otra parte Tim Bailey y Hugh Durrant como autores del artículo *“Simultaneous Localization and Mapping (SLAM): Part II”*[2] definen el SLAM como el proceso de un robot móvil que puede construir un mapa y al mismo tiempo usar este mapa para calcular su ubicación, en el SLAM en los últimos años se ha enfocado en mejorar la parte del cómputo garantizando el tiempo de eficiencia de los algoritmos para la estimación de la pose(orientación y ubicación) del robot móvil, según los autores propones tres aspectos fundamentales del SLAM, la complejidad computacional, la asociación de datos y la representación del ambiente o del entorno en el que se encuentra el robot móvil.

En la actualidad se han desarrollado diferentes técnicas propuestas para la localización y reconstrucción simultanea de los entornos por su parte Fernando A. Auat Cheeín, Fernando di Sciascio, Ricardo Carelli en su artículo *“Planificación de Caminos y Navegación de un Robot Móvil en Entornos Gaussianos mientras realiza tareas de SLAM”*[3] definen los algoritmos del SLAM que están basados en la extracción de características del entorno, también dicen que SLAM depende mucho del sensor que se utilice, sensores de rango para poder extraer segmentos o vértices o si se utiliza una cámara la identificación colores o patrones que identifican a cada característica; los anteriores autores implementan el filtro de kalman extendido y en la extracción de segmentos y esquinas

como características gaussianas del ambiente. Este filtro de kalman extendido se utiliza cuando las mediciones que se obtienen del robot no son lineales o para sistemas de dinámica no lineal, por lo que ellos proponen es tomar los puntos donde la incertidumbre sea mayor, girando el robot móvil 360° y luego de llegar al punto de mayor incertidumbre vuelve a repetir esta misma acción, cuando el agente robótico detecta un obstáculo se genera un destino alternativo, pero como no pretende perder el objetivo inicial el recorrido queda formado por el segmento robot – Destino de Desvió más el segmento Robot – punto de máxima incertidumbre.

La localización es el tema más importante a la hora de hablar del SLAM y como lo define Azucena Fuentes Ríos en su proyecto de grado "LOCALIZACIÓN Y MODELADO SIMULTÁNEOS EN ROS PARA LA PLATAFORMA ROBÓTICA MANFRED." [4] La localización es el problema de estimar la pose (orientación y ubicación) del robot en relación con un mapa, la pregunta más difícil que se debe cuestionar el robot es "¿Dónde estoy?", la localización puede ser local cuando se conoce la pose inicial del robot y global cuando no se conoce esta pose inicial, e indican los principales problemas que se tienen a la hora de construir el mapa:

- Donde guiar el robot durante la exploración.
- Detectar el ruido generado por la pose y las observaciones.
- Lidar la incertidumbre en el modelo y la forma de interpretar los datos de los sensores.
- Como modelar los cambios en el entorno a través del tiempo.

En consecuencia con el anterior documento mencionan los tipos de mapas y como se maneja el SLAM en ellos; uno de ellos es el SLAM topológico el cual el robot explora el entorno siguiendo una serie de criterios con los cuales determina una trayectoria y al mismo tiempo va guardando la descripción de los lugares, en el también nombran los mapas de ocupación de celdilla y mapa de características.

Como se nombró anteriormente el SLAM también se puede realizar por medio de una cámara para la detección de características en específico, de aquí aparece el termino odometría, de acuerdo con Roberto García García en su proyecto de final de carrera "Sistema de Odometría Visual para la Mejora del Posicionamiento Global de un Vehículo" [5] define la odometría visual como el proceso mediante el cual se determina la posición y la orientación de una cámara o de un sistemas de cámaras mediante una secuencia de imágenes adquiridas sin ningún

conocimiento previo del entorno, como el movimiento de las cámaras es el mismo que el del robot móvil, esta técnica también es conocida como “ego-motion”, también otra parte de la odometría se basa en encoders y en sensores exteroceptivos que por medio de estos se puede saber la velocidad y la distancia recorrida del robot móvil pero en este aspecto de la odometría se tiene varios errores sistemáticos con respecto a las ruedas.

1. AGENTE ROBOTICO

El agente que se utilizó es un robot tipo diferencial, es decir que no tiene ruedas directrices, la dirección del agente está dada por la velocidad relativa de las ruedas traseras. La rueda delantera, es una rueda de castor para mantener el balance del robot móvil.

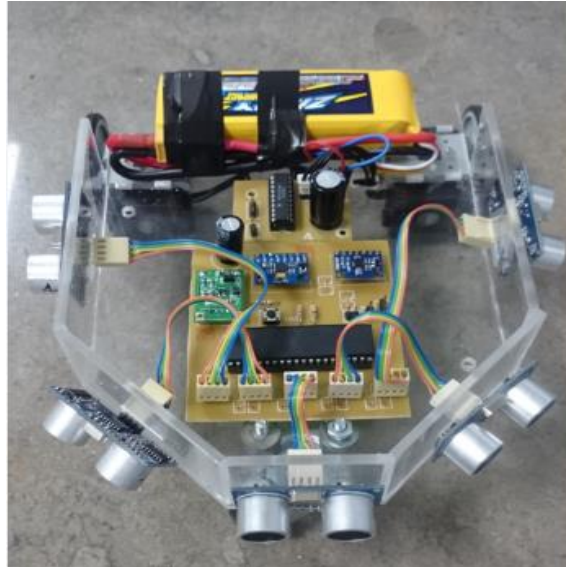


Imagen 1 Robot Diferencial

Fuente: Autor

Para el diseño del agente robótico se tuvieron en cuenta los siguientes parámetros de diseño [6]:

Maniobrabilidad: La maniobrabilidad de un robot móvil diferencial, puede ser una desventaja cuando el movimiento que realiza el robot es en línea recta, el giro de estos vehículos si depende del control que se realice a las dos ruedas traseras, pero este puede girar sobre sí mismo, lo que lo hace útil para desplazarse sobre entornos con obstáculos.

Controlabilidad: El control para este tipo de locomoción, es muy complejo por las diferencias que existen de las ruedas traseras, como las diferencias de fricción, diámetro y de control puesto que cada rueda es independiente y pueden girar a diferentes velocidades.

Estabilidad: Para la parte de la estabilidad del agente robótico se tuvieron en cuenta los siguientes errores:

- Desplazamiento en suelos desnivelados

- Patinaje de las ruedas por suelos resbaladizos y rotación excesivamente rápida.
- Mal alineamiento de las ruedas

Por los errores anteriores el robot se diseñó para terrenos nivelados y controlado a una velocidad considerable para que su rotación sea lenta para evitar volcamientos.

Incertidumbre: para poder lograr la mejor eficiencia del robot móvil se deben tener en cuenta los siguientes aspectos:

- Tener seguridad en el desplazamiento para evitar colisiones con obstáculos.
- calibración y control de los sensores, motores y cámara.
- Calidad del movimiento es decir que no hayan movimientos bruscos durante la ejecución de su recorrido.

Un aspecto muy importante de los robots móviles es que su naturaleza es no holonómica [7], la holonomicidad es una característica que depende de la movilidad del robot, se pueden clasificar dos grupos, en los holonómicos están todos los robots que tiene extremidades como brazos y manipuladores, y los no holonómicos en lo que se encuentran los robots móviles.

Los grados de libertad (DOF) de los robots móviles se encuentran en dos grupos los robots terrestres son de 2 DOF y los robots aéreos y acuáticos si poseen los 3 DOF [7], al tener 2 grados de libertad indica que tiene una rotación y dos traslaciones y al tener 3 grados de libertad indica que tiene tres rotaciones y tres traslaciones.

1.1 Modelamiento cinemático

El modelamiento cinemático de un robot móvil tipo diferencial se puede considerar como un conjunto de cadenas cinemáticas cerradas, cuyo número está dado por la cantidad de ruedas que toquen el suelo.

Para la construcción del sistema se consideran las siguientes limitaciones:

- El robot se mueve sobre una superficie plana.
- No existe elementos flexibles en la estructura del robot

- Se considera que las ruedas poseen un eje de direccionamiento, que siempre es perpendicular al suelo.
- Se desprecia todo tipo de fricción en los elementos móviles del vehículo contra el suelo.

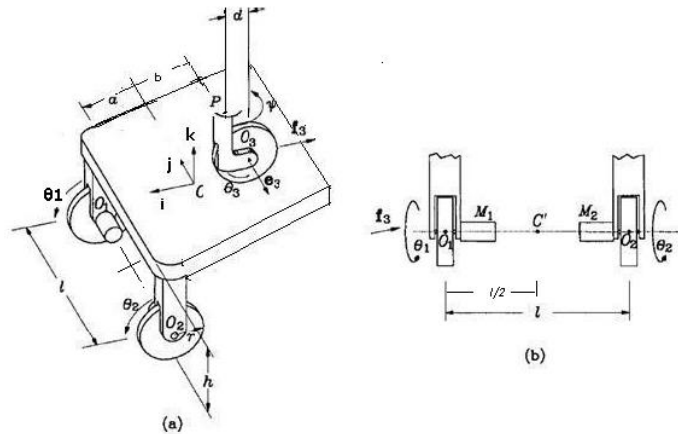


Imagen 2 Esquema del robot

Fuente: MODELO CINEMÁTICO DINÁMICO DEL MINI ROBÓT MÓVIL RICIMAF, 2012.p 51

Para poder resolver el modelo cinemático del robot, el cual en la navegación va a realizar cambios diferenciales tanto en su posición como en su orientación y para poder hallarlos se realizó mediante la matriz jacobiana y su inversa.

Parámetros del robot:

- a** distancia entre el centro del chasis y los centros de las ruedas.
- b** distancia entre el centro del chasis y el centro del soporte de la rueda castor.
- d** distancia entre el centro del soporte de la rueda loca y el centro de la rueda loca.
- l** distancia entre los centros de las ruedas traseras.
- h** alturas del chasis sobre el terreno.

r1 radio de las ruedas traseras.

r2 radio de la rueda castor.

Ψ ángulo de giro del chasis con respecto al sistema de coordenada en el chasis del robot $\{i,j\}$.

θ_i , θ_d , θ_l ángulos de giro de las ruedas del sistema sobre su eje respectivamente.

Las ecuaciones cinemáticas de posición surgen al diferenciar con respecto al tiempo las ecuaciones de posición.

$$\dot{y} = v \cos \Psi \quad (1)$$

$$\dot{x} = -v \sin \Psi \quad (2)$$

Donde v es la velocidad lineal del chasis y su velocidad angular está dada por:

$$\dot{\Psi} = W \quad (3)$$

La ecuación anterior representa la ecuación diferencial de la orientación del robot, ya obteniendo estos tres valores se puede establecer un sistema de coordenadas, como lo es el vector de localización en coordenadas globales y se obtiene también el vector tanto de la velocidad lineal como de la velocidad angular, peor este segundo vector si está en las coordenadas generalizadas del robot.

$$\dot{p} = [\dot{x} \quad \dot{y} \quad \dot{\Psi}]^T \quad (4)$$

$$\dot{q} = [v \quad w]^T \quad (5)$$

El modelo diferencial de forma matricial es:

$$\dot{p} = J(p)\dot{q} \quad (6)$$

Donde $J(p)$ es una matriz de derivadas parciales o jacobiana. A partir de las ecuaciones (1) y (2) se pueden expresar en forma matricial y de esta manera obtener los elementos de la matriz jacobiana.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} -\sin\Psi & 0 \\ \cos\Psi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \quad (7)$$

Las variables de velocidad lineal y angular representan variables de entrada o de control, por esta razón la matriz jacobiana queda de la siguiente manera.

$$J(p) = \begin{bmatrix} -\sin\Psi & 0 \\ \cos\Psi & 0 \\ 0 & 1 \end{bmatrix} \quad (8)$$

Para obtener las ecuaciones cinemáticas diferenciales inversas es necesario invertir la matriz anterior, pero esta es una matriz singular por tanto es necesario obtener su pseudoinversa a partir de ecuación (6), así que se multiplica por la transpuesta de la matriz jacobiana en ambos lados de la ecuación:

$$J(p)^T \dot{p} = J(p)^T \dot{q} J(p) \quad (9)$$

Despejando \dot{q} :

$$\dot{q} = J(p)^T \dot{p} \quad (10)$$

De tal manera que el modelo cinemático diferencial inverso del robot móvil expresado de forma matricial queda:

$$\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Psi} \end{bmatrix} \quad (11)$$

Las velocidades angulares y lineales de las ruedas traseras están definidas como:

$$v = \frac{(v_i + v_d)}{2} = \frac{(\theta_i + \theta_d) r}{2} \frac{1}{l} \quad (12)$$

$$w = \frac{(v_i - v_d)}{l} = \frac{(\theta_i - \theta_d) r}{l} \frac{1}{1} \quad (13)$$

Despejando θ_i y θ_d de las ecuaciones (12) y (13)

$$W_i = \frac{v - \frac{\theta_i}{2}}{2} \quad W_d = \frac{v + \frac{\theta_d}{2}}{2} \quad (14)$$

$$\dot{x} = \frac{(\theta_i + \theta_d)r}{2} \sin\Psi \quad (15)$$

$$\dot{y} = \frac{(\theta_i + \theta_d)r}{2} \cos\Psi \quad (16)$$

$$\dot{\Psi} = \frac{(\theta_i - \theta_d)r}{l} \quad (17)$$

Donde $r = 2.6$ y $l = 17$

Expresando en forma matricial.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} -1.3\sin\Psi \\ 1.3\cos\Psi \\ -0.152 \end{bmatrix} \theta_i + \begin{bmatrix} -1.3\sin\Psi \\ 1.3\cos\Psi \\ -0.152 \end{bmatrix} \theta_d \quad (18)$$

De la matriz anterior se obtiene el modelo cinemático diferencial directo del robot.

1.2 Restricciones cinemáticas.

1.2.1 Primera restricción

La primera restricción es sobre las ruedas traseras, es una restricción de tipo rodante, cantidad de movimiento a lo largo de la dirección del plano de la rueda izquierda o derecha sea igual, al giro de la rueda sobre su eje horizontal, con el fin de obtener un desplazamiento puro en el punto del contacto sin deslizamiento lateral.

Las expresiones matemáticas de esta restricción para cada rueda, separando las ruedas con un subíndice, para la rueda derecha la letra d y para la rueda izquierda la letra i.

$$[\dot{x}\cos\Psi + \dot{y}\sin\Psi] + \frac{1}{2}\dot{\Psi} = r\dot{\theta}_i \quad (19)$$

$$[\dot{x}\cos\Psi + \dot{y}\sin\Psi] - \frac{1}{2}\dot{\Psi} = r\dot{\theta}_d \quad (20)$$

1.2.2 Segunda restricción

La segunda restricción es de tipo deslizante, es decir que la rueda no debe resbalar ortogonal al plano de la rueda, por lo tanto la velocidad lineal quedaría como:

$$-\dot{x}\sin\Psi + \dot{y}\cos\Psi = v \quad (21)$$

Esta restricción indica que la fuerza en dirección ortogonal a la rueda debe ser 0.

1.3.3 Tercera restricción

Esta restricción dice que si existe movimiento lateral en las ruedas traseras por diferentes causas, la mal alineación de las ruedas y los suelos resbaladizos, por esta razón la expresión matemática quedaría de la siguiente manera.

$$-\dot{x}\sin\Psi + \dot{y}\cos\Psi = 0 \quad (22)$$

Las tres restricciones son de tipo no holonómico, porque estas dependen de las velocidades, por esta similitud de las restricciones uno y dos podemos obtener una sola restricción reduciendo estas dos ecuaciones y quedaría de la siguiente manera:

$$v = [\dot{x}\cos\Psi + \dot{y}\sin\Psi] = \frac{1}{2}r(\dot{\theta}_i + \dot{\theta}_d) \quad (23)$$

Restando las ecuaciones (21) y (22) y luego integrándola se elimina el término de la velocidad lineal y se obtendría de esta operación una restricción de tipo holonómico.

$$\dot{\Psi} = \frac{r}{l}(\dot{\theta}_i + \dot{\theta}_d) \quad \Psi = \frac{r}{l}(\theta_i + \theta_d) \quad (24)$$

$$\dot{x}\cos\Psi + \dot{y}\sin\Psi = \frac{1}{2}r(\dot{\theta}_i + \dot{\theta}_d) \quad (25)$$

Por lo tanto las dos restricciones no holonómicas serían las ecuaciones (22) y (25).

Las anteriores restricciones se pueden representar matricialmente de la siguiente manera:

$$A(q)\dot{q} = 0 \quad (26)$$

Donde:

$$q = [\dot{q}_1 \ \dot{q}_2 \ \dot{q}_3 \ \dot{q}_4]^T = [\dot{x} \ \dot{y} \ \dot{\theta}_i \ \dot{\theta}_d]^T \quad (27)$$

$$A(q) = \begin{bmatrix} \sin\Psi & -\cos\Psi & 0 & 0 \\ \cos\Psi & \sin\Psi & 0.5r & 0.5r \end{bmatrix} \quad (28)$$

1.3 Modelamiento Dinámico

El modelamiento dinámico del robot, representado por las ecuaciones del movimiento del agente robótico se realizó usando la formulación de Euler y LaGrange. Mostrada en la ecuación (7).

La energía total E de un robot de n DOF es la suma de las energías cinéticas y potenciales [7].

$$\mathcal{E}(q, \dot{q}) = \mathcal{K}(q, \dot{q}) + \mathcal{U}(q) \quad (29)$$

El Lagrangeano es la diferencia entre su energía cinética y potencial.

$$L(q, \dot{q}) = \mathcal{K}(q, \dot{q}) - \mathcal{U}(q) \quad (30)$$

Se asume que la energía potencial se debe solo a las fuerzas conservadoras, como la energía gravitacional, principalmente se determina el Lagrangeano del sistema como se expresó en la ecuación (30), la energía potencial del robot se anula debido a que la altura es cero, porque se considera el movimiento sobre un terreno plano y horizontal y la energía cinética depende de las masas y las velocidades.

- Masa de ruedas traseras energizadas = **mi = md = mr**
- Masa de la rueda loca **ml0**.
- Masa del chasis **mc**



Imagen 3 Masas de las Ruedas y el chasis

Fuente: Autor

Las velocidades que adquieren estas masas son [7]:

- Velocidad lineal de las ruedas traseras y rueda loca **v_i , v_d , v_{l0}** .
- Velocidad rotacional de las ruedas alrededor de su eje **$\dot{\theta}_i$, $\dot{\theta}_d$, $\dot{\theta}_{l0}$** .
- Velocidad Angular de las ruedas **w_i , w_d , w_{l0}** .
- Velocidad lineal del centro del chasis o plataforma **v_c**
- Velocidad angular del centro del chasis **W**

1.3.1 Velocidades de las ruedas.

Las velocidades lineales de los centros de las ruedas son:

$$v_i = r\dot{\theta}_i; v_d = r\dot{\theta}_d; v_{l0} = -r\dot{\theta}_{l0} \quad (31)$$

La velocidad del centro del chasis es:

$$\sum_{K=1}^2 [V_{ck} = r\dot{\theta}_k + (C - O_k)\dot{\Psi}] \quad (32)$$

Restando los términos de la ecuación (32)

$$0 = r(\dot{\theta}_i - \dot{\theta}_d) - (O_1 - O_2)\dot{\Psi} \quad (33)$$

Se obtiene la velocidad angular del centro del chasis w .

$$W = \dot{\Psi} = \frac{r}{l}(\dot{\theta}_i - \dot{\theta}_d) \quad (34)$$

Si se suman los términos de la ecuación (32)

$$V_c = \frac{r}{2}(\dot{\theta}_i - \dot{\theta}_d) + a\dot{\Psi} \quad (35)$$

La velocidad lineal del centro del chasis V_c es:

$$V_c = a\frac{r}{l}(\dot{\theta}_i - \dot{\theta}_d)\mathbf{i} + \frac{r}{2}(\dot{\theta}_i - \dot{\theta}_d)\mathbf{j} \quad (36)$$

Las ecuaciones (34) y (36) constituyen la cinemática directa diferencial del robot.

La velocidad angular de las ruedas energizadas.

$$W_i = \dot{\theta}_i\mathbf{i} + \dot{\Psi}\mathbf{k} = \dot{\theta}_i\mathbf{i} + \frac{r}{l}(\dot{\theta}_i - \dot{\theta}_d)\mathbf{k} \quad (37)$$

$$W_d = \dot{\theta}_d\mathbf{i} + \dot{\Psi}\mathbf{k} = \dot{\theta}_d\mathbf{i} + \frac{r}{l}(\dot{\theta}_i - \dot{\theta}_d)\mathbf{k} \quad (38)$$

La velocidad angular de la rueda castor es:

$$W_{lo} = -\dot{\theta}_{lo}\mathbf{f}_3 + (W_c + \dot{\sigma}) \quad (39)$$

$$W_{lo} = -\dot{\theta}_{lo}\mathbf{f}_3 + \frac{r}{l}(\dot{\theta}_i - \dot{\theta}_d) + \dot{\sigma} \quad (40)$$

Para poder determinar la velocidad angular de la rueda loca es necesario conocer los valores de $\dot{\theta}_{lo}$ y $\dot{\sigma}$, que son las velocidades de rotación y el ángulo formado entre los vectores unitarios \mathbf{j} y \mathbf{e}_3 de los sistemas de

coordenadas del centro del chasis y el soporte de la rueda loca, el punto **P** es el centro del soporte de la rueda loca y su velocidad es W_s , expresando en los parámetros del soporte [7]:

$$\dot{P} = V_{lo} + W_s(P + O_{lo}) \quad (41)$$

Como:

$$V_{lo} = -r\dot{\theta}_{lo}f_3; \quad W_s = W_c + \dot{\sigma}(P + O_{lo})de_3e_3 \quad (42)$$

$$\dot{P} = -r\dot{\theta}_{lo}f_3 + de_3e_3(W_c + \dot{\sigma}) \quad (43)$$

Representando la velocidad P en función de los parámetros del chasis.

$$\dot{P} = V_c + bW_{ci} \quad (44)$$

Cuando se iguala la ecuación (43) y (44)

$$-r\dot{\theta}_{lo}f_3 + de_3(W_c + \dot{\sigma}) = V_c + bW_{ci} \quad (45)$$

$$-r\dot{\theta}_{lo}f_3 + de_3\dot{\sigma} = V_c + bW_{ci} - de_3W_c \quad (46)$$

Los vectores unitarios de los mismos sistemas de coordenadas permite obtener los valores V_{lo} y $\dot{\sigma}$ [7] :

$$e_3 \cdot e_3 = f_3 \cdot f_3 = 1; \quad e_3 \cdot f_3 = f_3 \cdot e_3 = 0 \quad (47)$$

Al multiplicar escalarmente f_3 a ambos lados de la ecuación (46) se obtiene $\dot{\theta}_{lo}$:

$$\dot{\theta}_{lo} = -\frac{1}{r}[V_c \cdot f_3 + bW_{ci} \cdot f_3] \quad (48)$$

Y se halló $\dot{\sigma}$ multiplicando escalarmente e_3 a ambos lados de la ecuación (41)

$$\dot{\sigma} = \frac{1}{d} [V_c \cdot e_3 + bW_{ci} \cdot e_3 + dW_c] \quad (49)$$

Se realiza la transformación de coordenadas ya que el centro del chasis se encuentra en $\{i,j\}$ y el sistema de coordenadas del a rueda castor es $\{e_3,f_3\}$ la relación entre estos dos sistemas de coordenadas está dada por:

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} -\sin\varphi & -\cos\varphi \\ \cos\varphi & -\sin\varphi \end{bmatrix} \begin{bmatrix} e_3 \\ f_3 \end{bmatrix} \quad (50)$$

$$i \cdot e_3 = -\sin\varphi; \quad i \cdot f_3 = -\cos\varphi; \quad (51)$$

Trasformando la ecuación (36).

$$V_c = a \frac{r}{l} (\dot{\theta}_i - \dot{\theta}_d) (-\sin\varphi e_3 - \cos\varphi f_3) + \frac{r}{2} (\dot{\theta}_i - \dot{\theta}_d) (\cos\varphi e_3 - \sin\varphi f_3) \quad (52)$$

$$V_c = \left[a \frac{r}{l} (\dot{\theta}_i - \dot{\theta}_d) \sin\varphi + \frac{r}{2} (\dot{\theta}_i - \dot{\theta}_d) \cos\varphi \right] e_3 - \left[a \frac{r}{l} (\dot{\theta}_i - \dot{\theta}_d) \cos\varphi + \frac{r}{2} (\dot{\theta}_i - \dot{\theta}_d) \sin\varphi \right] f_3 \quad (53)$$

Lo que se hace a continuación es coger la ecuación (53) multiplicar a ambos lados por f_3 , luego se sustituye esta y la ecuación (51) estas dos en la ecuación (48) y se obtiene el valor de la velocidad de rotación de la rueda de castor.

$$\dot{\theta}_{lo} = \left[\frac{(a+b)}{l} \cos\varphi (\dot{\theta}_i - \dot{\theta}_d) + \frac{1}{2} (\dot{\theta}_i - \dot{\theta}_d) \sin\varphi \right] \quad (54)$$

Ahora se multiplico por e_3 la ecuación (53) y sustituyendo esta ecuación y la ecuación (51) en la ecuación (49) se obtiene el valor de la velocidad angular de la rueda loca.

$$\dot{\sigma} = \left[- \left(\frac{r(a+b)}{dl} \sin\varphi - \frac{r}{l} \right) (\dot{\theta}_i - \dot{\theta}_d) + \frac{r}{2d} (\dot{\theta}_i - \dot{\theta}_d) \cos\varphi \right] \quad (55)$$

Ahora para obtener la velocidad angular de la rueda castor en función de las variables de entrada sustituimos las ecuaciones (54) y (55) en la ecuación (40).

$$W_{lo} = \left[\frac{r}{l} - \frac{a+b}{l} \cos\varphi - \frac{r(a+b)}{dl} \sin\varphi \right] (\dot{\theta}_i - \dot{\theta}_d) + \left[\frac{r}{2d} \cos\varphi - \frac{1}{2} \cos\varphi \right] (\dot{\theta}_i - \dot{\theta}_d) \quad (56)$$

1.3.2 Energías Cinéticas

Ya definido las masas y velocidades, se calculó las Energías cinéticas correspondientes.

$$k = k_T + k_R + k_C \quad (57)$$

Donde **KT** es la energía trasnacional, **KR** es la energía rotacional y **Kc** es la aportación de la energía cinética del centro de masas de gravedad de todo el robot al punto central [7], ya que este punto puede no coincidir donde ocurre la intersección del eje de simetría del sistema con el eje horizontal el cual giran las dos ruedas y la velocidad queda definida por las velocidades debidas a las restricciones [7].

$$K_T = \frac{1}{2} (M + m_c) V_c^2 \quad (57)$$

Donde:

$$V_c^2 = \dot{x}_c^2 + \dot{y}_c^2 \quad (59)$$

Donde **M** es la suma total de cada una de las masas de las ruedas y m_c es la masa de chasis sin ruedas.

$$k_R = [I_{ri} W_i^2 + I_{rd} W_d^2 + I_{rlo} W_{lo}^2] + \frac{1}{2} I_c \dot{\Psi}^2 \quad (60)$$

I_r Expresa el momento de inercia y \mathbf{W} la velocidad angular de las ruedas alrededor de los ejes horizontales de las ruedas.

$$I_{ri} = m_i r^2; \quad I_{rd} = m_d r^2; \quad I_{rlo} = m_{lo} r^2 \quad (61)$$

En la ecuación anterior la r es el radio de cada rueda, el siguiente paso que se realizó es que el momento de inercia en el centro del chasis es $I_c = m_c z$ con respecto a un eje vertical que se levanta en la intersección del eje de simetría y eje horizontal de las ruedas motrices. Tomando las ecuaciones (24) y (25) se obtiene la contribución de la energía cinética del centro de la masa dada por el producto de la masa del chasis m_c y por lo tanto z es el producto de las velocidades de restricción cinemática [7].

$$z = \frac{r}{l} (\dot{\theta}_i - \dot{\theta}_d) (\dot{x} \sin \Psi - \dot{y} \cos \Psi) \quad (62)$$

$$k_c = \frac{1}{2} m_c z = \frac{1}{2} m_c \frac{r}{l} (\dot{\theta}_i - \dot{\theta}_d) (\dot{x} \sin \Psi - \dot{y} \cos \Psi) \quad (63)$$

$$k_c = m_c \frac{r}{2l} [(\dot{x} \sin \Psi - \dot{y} \cos \Psi) \dot{\theta}_i - (\dot{x} \sin \Psi - \dot{y} \cos \Psi) \dot{\theta}_d] \quad (64)$$

Representa la velocidad angular de la masa del chasis en la ecuación (34), para que las expresiones queden en función de las velocidades de la rueda usando la ecuación (36) en (58) queda de la siguiente manera:

$$k_R = M(\dot{x}_c^2 + \dot{y}_c^2) + m_c \left[a \frac{r}{l} (\dot{\theta}_i - \dot{\theta}_d) \mathbf{i} + \frac{r}{2} (\dot{\theta}_i - \dot{\theta}_d) \mathbf{j} \right]^2 \quad (65)$$

$$k_R = M(\dot{x}_c^2 + \dot{y}_c^2) + m_c r^2 \left[\left(\frac{a}{l} \right)^2 (\dot{\theta}_i - \dot{\theta}_d)^2 + \frac{1}{4} (\dot{\theta}_i - \dot{\theta}_d)^2 \right] \quad (66)$$

Usando las ecuaciones (34) (37) (38) (54) (60)

$$k_R = \frac{1}{2} [I_{ri} W_i^2 + I_{rd} W_d^2 + I_{rlo} W_{lo}^2] + \frac{1}{2} I_c \dot{\Psi}^2 = \quad (67)$$

$$\begin{aligned}
& \frac{1}{2} I_r \left[\dot{\theta}_i^2 + \left(\frac{a}{l} \right)^2 (\dot{\theta}_i - \dot{\theta}_d)^2 + \dot{\theta}_d^2 + \left(\frac{r}{l} \right)^2 (\dot{\theta}_i - \dot{\theta}_d)^2 \right] + \frac{1}{2} I_{rlo} \\
& \frac{1}{2} I_{rlo} \left[\left(\frac{a+b}{l} \right) \cos \psi (\dot{\theta}_i - \dot{\theta}_d) + \frac{1}{2} (\dot{\theta}_i - \dot{\theta}_d) \sin \psi \right]^2 + \frac{1}{2} I_c \left[\frac{r}{l} (\dot{\theta}_i - \dot{\theta}_d) \right]^2 \\
& k_R = \frac{1}{2} I_r \left[(\dot{\theta}_i^2 + \dot{\theta}_d^2) + 2 \left(\frac{r}{l} \right)^2 (\dot{\theta}_i - \dot{\theta}_d)^2 \right] + \quad (68)
\end{aligned}$$

$$\begin{aligned}
& \frac{1}{2} I_{rlo} \left[\left(\frac{a+b}{l} \right)^2 \cos^2 \psi (\dot{\theta}_i - \dot{\theta}_d)^2 + \left(\frac{1}{4} \right) \sin^2 \psi (\dot{\theta}_i - \dot{\theta}_d)^2 \right. \\
& \left. + \left(\frac{a+b}{l} \right) \sin \psi \cos \psi (\dot{\theta}_i - \dot{\theta}_d) (\dot{\theta}_i + \dot{\theta}_d) \right] + \frac{r^2}{2l^2} I_c (\dot{\theta}_i - \dot{\theta}_d)^2
\end{aligned}$$

En robótica móvil la ecuación de Euler-Lagrange es general para cualquier robot y está en función del Lagrangeano y de las coordenadas generalizadas del robot:

$$\tau = \frac{\partial}{\partial t} \left[\frac{\partial}{\partial \dot{q}} (L(q, \dot{q})) \right] - \frac{\partial}{\partial q} [L(q, \dot{q})] \quad (69)$$

Derivando las expresiones de la energía cinética con respecto a las velocidades de las ruedas del agente robótico, se obtuvo esto mediante la ecuación de Euler – LaGrange, recordando que la energía potencial es constante cuando el robot móvil se está desplazando.

$$\frac{d}{dt} \left[\frac{\partial}{\partial \dot{q}} [K] \right] = \frac{d}{dt} \left[\frac{\partial}{\partial \dot{q}} (k_T + k_R + k_C) \right] \quad (70)$$

$$T_1 = \frac{\partial k_T}{\partial \dot{q}} = 2(\dot{x}_2 + \dot{y}_2) + m_c r^2 (\dot{\theta}_i + \dot{\theta}_d) \quad (71)$$

$$T_2 = \frac{\partial k_R}{\partial \dot{q}} = I_r(\dot{\theta}_i + \dot{\theta}_d) \quad (72)$$

$$+ I_{rlo} \left\{ \frac{1}{2} \sin^2 \Psi (\dot{\theta}_i + \dot{\theta}_d) \right. \\ \left. + \left(-\frac{1}{2} \left(\frac{a+b}{l} \right) \right) \sin \Psi \cos \Psi (\dot{\theta}_i + \dot{\theta}_d) \right. \\ \left. + \left(\frac{a+b}{l} \right) \sin \Psi \cos \Psi (\dot{\theta}_i - \dot{\theta}_d) \right\}$$

$$T_3 = \frac{\partial k_C}{\partial \dot{q}} = m_c \frac{r}{l} [(\sin \Psi - \cos \Psi) \dot{\theta}_i + (\cos \Psi - \sin \Psi) \dot{\theta}_d] \quad (73)$$

El primer término de la ecuación de Euler-LaGrange es la diferenciación de los términos.

$$\frac{d}{dt} \frac{\partial (k_T + k_R + k_C)}{\partial \dot{q}} \quad (74)$$

Derivando con respecto a tiempo las ecuaciones quedan de la siguiente manera.

$$\frac{dT_1}{dt} = 2(\ddot{x}_c + \ddot{y}_c) + m_c r^2 (\ddot{\theta}_i + \ddot{\theta}_d) \quad (75)$$

$$\frac{dT_2}{dt} = I_r(\ddot{\theta}_i + \ddot{\theta}_d) \quad (76) \\ + I_{rlo} \left[\left(\frac{a+b}{l} \right) \sin \Psi \cos \Psi (\ddot{\theta}_i - \ddot{\theta}_d) \right. \\ + \left(\frac{1}{2} \right) \sin^2 \Psi (\ddot{\theta}_i + \ddot{\theta}_d) \\ + \left(\frac{a+b}{l} \right) (\cos^2 \Psi - \sin^2 \Psi) (\dot{\theta}_i - \dot{\theta}_d) \dot{\Psi} \\ \left. + \sin \Psi \cos \Psi (\dot{\theta}_i + \dot{\theta}_d) \dot{\Psi} \right]$$

$$\frac{dT_3}{dt} = m_c \frac{r}{l} \left[(\sin\Psi + \cos\Psi)\dot{\Psi}\dot{\theta}_i + (\sin\Psi - \cos\Psi)\dot{\Psi}\dot{\theta}_d + (\sin\Psi - \cos\Psi)\ddot{\theta}_i - (\sin\Psi + \cos\Psi)\ddot{\theta}_d \right] \quad (77)$$

Representando los términos anteriores de forma matricial.

$$M(q)\ddot{q} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \quad (78)$$

$$\dot{M}(q)\dot{q} = \begin{bmatrix} m_{15} \\ m_{25} \\ m_{35} \\ m_{45} \end{bmatrix} \quad q = \begin{bmatrix} x_c \\ y_c \\ \theta_i \\ \theta_d \end{bmatrix} \quad (79)$$

Donde estos valores quedan indicados como coeficientes no lineales del vector de aceleración generalizado $\ddot{\theta}$. y de la velocidad lineal \dot{v} .

$$m_{11} = 2\ddot{x}_c; \quad m_{12} = 2\ddot{y}_c; \quad m_{13} = m_c r^2 \ddot{\theta}_i; \quad (80)$$

$$m_{14} = m_c r^2 \ddot{\theta}_d; \quad m_{15} = 0$$

$$m_{21} = 0; \quad m_{22} = 0; \quad (81)$$

$$m_{23} = \left\{ I_r + I_{rlo} \left[\left(\frac{a+b}{l} \right) \sin\Psi \cos\Psi + \left(\frac{1}{2} \right) \sin^2\Psi \right] \right\} \ddot{\theta}_i$$

$$m_{24} = \left\{ -I_r + I_{rlo} \left[-\left(\frac{a+b}{l} \right) \sin\Psi \cos\Psi + \left(\frac{1}{2} \right) \sin^2\Psi \right] \right\} \ddot{\theta}_d$$

$$m_{25} = I_{rlo} \left(\frac{a+b}{l} \right) (\cos^2\Psi - \sin^2\Psi) + (\dot{\theta}_i - \dot{\theta}_d)\dot{\Psi} + I_{rlo} \sin\Psi \cos\Psi (\dot{\theta}_i + \dot{\theta}_d)\dot{\Psi}$$

$$m_{31} = 0; \quad m_{32} = 0; \quad m_{33} = -m_c \frac{r}{l} (\cos\Psi - \sin\Psi) \ddot{\theta}_i; \quad (82)$$

$$m_{33} = -m_c \frac{r}{l} (\cos\Psi - \sin\Psi) \ddot{\theta}_d;$$

$$m_{45} = 0 \quad (83)$$

Se calculan las derivadas parciales de la energética cinética con respecto al vector posición.

$$\frac{1}{2} \left[\frac{\partial}{\partial t} (k_T + k_R + k_C) \right] \quad (84)$$

$$\frac{\partial k_T}{\partial \dot{q}} = 0; \quad \frac{\partial k_R}{\partial \dot{q}} = 0; \quad \frac{\partial k_C}{\partial \dot{q}} = 0; \quad (85)$$

Para terminar se tomó los coeficientes de Euler – LaGrange y las restricciones no holonómicas y quedan representadas de la siguiente manera:

$$\tau = M(q)\ddot{q} + C(q)\dot{q} + A^T(q)\lambda \quad (86)$$

2. Sensores, Actuadores y diseño del agente robótico

En este capítulo se presentaran los sensores utilizados por el agente robótico, se pueden clasificar de dos maneras para el caso de los robots móviles, los sensores propioceptivos los cuales indican el estado interno del robot, como lo son los giroscopios, acelerómetros y brújulas, y los sensores exteroceptivos los cuales se refieren a la percepción de aspectos externos al robot, como por ejemplo sensores de ultrasonido, presión y temperatura [8].

También se muestra los actuadores que se usaron para el agente móvil y toda su configuración y adaptación con la tarjeta que recibe todo los datos que en este caso es una Raspberry pi 2 model b

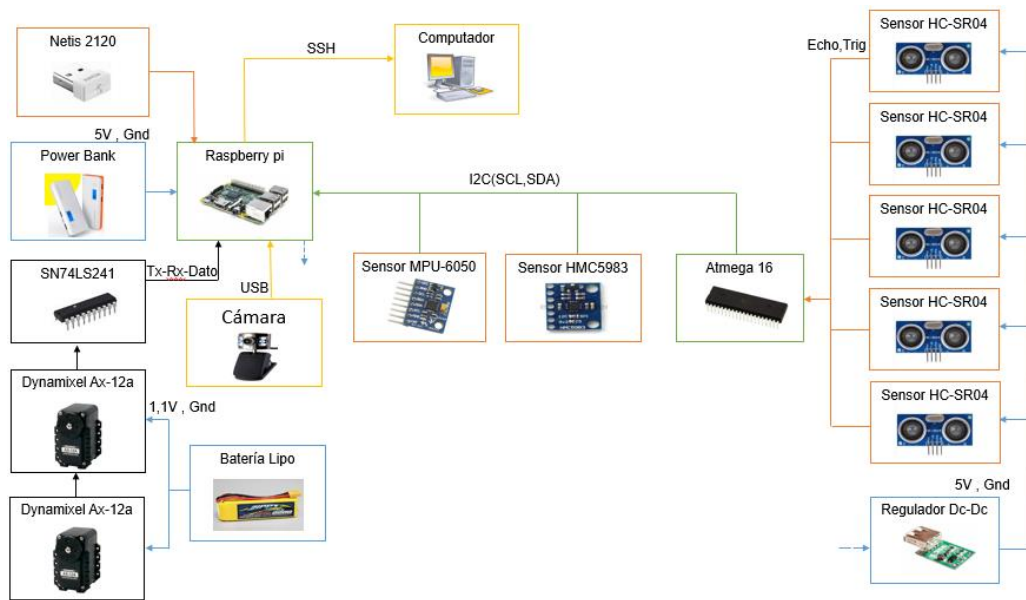


Diagrama 1 Componentes del agente robótico

Fuente: Autor

En el diagrama anterior se muestran todos los componentes que tiene el agente robótico, a continuación se muestra el funcionamiento de cada uno de estos componentes y la configuración que se realizó a cada uno de estos componentes.

1.1 Raspberry pi 2 Model b

Raspberry pi es un ordenador de placa reducida de bajo costo, con el objetivo de estimular la enseñanza de la ciencia de la computación, esta tarjeta cuenta con un sistema operativo llamado Raspbian Jessie [9], la instalación y configuración de este sistema operativo, se puede ver en el anexo (A).

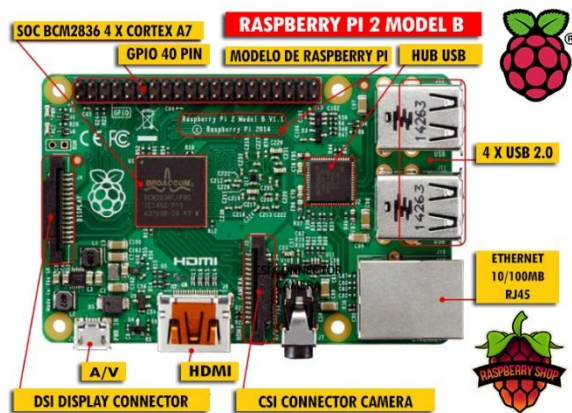


Imagen 4 Partes Raspberry pi model B

Fuente: raspberrypi.org. HARDWARE RASPBERRY PI. Recuperado 2 de agosto 2016, de: <http://www.raspberrypi.org/hardware-raspberrypi.php>

Tabla 1 Especificaciones técnicas de la Raspberry pi 2 model B

	RASPBERRY PI 2 MODEL B
SoC	BROADCOM BCM2836
CPU	ARM11 ARMv7 ARM CORTEX-A7 4 NUCLEOS 900Mhz
GPU	BROADCOM VIDEOCORE IV 250Mhz OPENGL 2.0
Memoria RAM	1Gb LPDDR2 SDRAM 450Mhz
Puertos USB	4
GPIO	40 pines
Video	HDMI 1.4 1920 X 1200
Almacenamiento	MICROSD

Ethernet 10/100MBPS	SI TIENE
Tamaño	85.6 X 56.5
Masa en gr	45

Fuente: raspberrysshop. HARDWARE RASPBERRY PI. Recuperado 2 de agosto 2016, de: <http://www.raspberrysshop.es/hardware-raspberry-pi.php>

1.2 Sensor HC-SR04

Los sensores de ultrasonido funcionan o se activan cuando se emite una señal ultrasónica en forma de pulso, para posteriormente recibir el reflejo o el eco de la señal enviada, para estos sensores es necesario tener un tiempo en blanco pequeño para que no se confunda con señal del emisor dando como medida mínima unos 2cm.

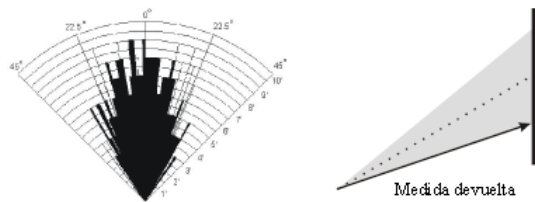


Imagen 5 Angulo de medida

Fuente: Instituto de Investigación Tecnológica. Sensores para robot móviles. Recuperado el 16 de agosto 2016, de: <http://www.iit.comillas.edu/~alvaro/teaching/Clases/Robots/teoria/Sensores%20y%20actuadores.pdf>



Imagen a

Imagen b

Imagen 6 : Imagen **a** diafonía , Imagen **b** Reflexión

Fuente: Instituto de Investigación Tecnológica. Sensores para robot móviles. Recuperado el 16 de agosto 2016, de: <http://www.iit.comillas.edu/~alvaro/teaching/Clases/Robots/teoria/Sensores%20y%20actuadores.pdf>

Este sensor tiene tres inconvenientes el primero es el ángulo de medida que varía, entre más alto sea la frecuencia más direccional es la onda pero se atenúa más, el segundo son las reflexiones, como se muestra en la imagen 6(b) el eco se refleja en una dirección diferente a la de retorno hacia el receptor, no se puede asegurar que la señal de eco sea un resultado de reflexiones en el entorno y la tercera es que si un sensor emite una señal el eco de esta señal no sea recibido por otro sensor [8]. La programación que se utilizó para este sensor se realizó en el atmega 16 se puede ver en el anexo (B) y la programación que se uso para leer los valores en la raspberry están en el anexo (I)

1.3 Sensor MPU6050

Es un sensor con 6 grados de libertad, este sensor lleva un acelerómetro y un giroscopio ambos de 3 ejes, este sensor utiliza un protocolo de comunicación I2C [10].

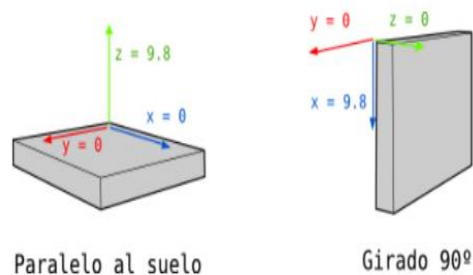


Imagen 7 Orientación Sensor MPU650

Fuente: Robologs. Tutorial de Arduino y MPU-6050. Recuperado 15 de agosto 2016, de: <http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/>

En la parte del acelerómetro, se puede medir en tres coordenadas partiendo de la aceleración de la tierra aproximadamente 9.8m/s^2 y con

esta medida se puede determinar la inclinación del sensor partiendo de las siguientes ecuaciones [10].

$$\text{Angulo Y} = \text{atan}\left(\frac{x}{\sqrt{y^2 + z^2}}\right) \quad (87)$$

$$\text{Angulo X} = \text{atan}\left(\frac{y}{\sqrt{x^2 + z^2}}\right) \quad (88)$$

En la parte del giroscopio, este mide la velocidad angular, si se determina el ángulo inicial, mediante la siguiente formula se obtiene el ángulo en **X** y **Y** del sensor.

$$\text{Angulo Y} = \text{Angulo Y anterior} + \text{GiroscopioY} * \Delta t \quad (89)$$

Donde él Δt es el tiempo de muestreo del sensor [10], la librería utilizada está en el anexo (C).

1.4 Sensor HMC5983

Este sensor es un magnetómetro de 3 ejes con compensación de temperatura, este sensor se comunica mediante el protocolo de I2C y con este sensor podemos determinar la dirección a la cual el robot móvil se dirige [11], la librería utilizada se encuentra en el anexo (D).

1.5 Motores Dynamixel Ax-12a

Son servomotores de alto desempeño los cuales son controlados mediante un protocolo de comunicación llamado serial asincrónica semiduplex, Estos motores se les adapto un integrado SN74LS241 ya que estos motores no responden una señal PWM, este integrado nos permite pasar de full-duplex a half-duplex para poder controlar estos motores.

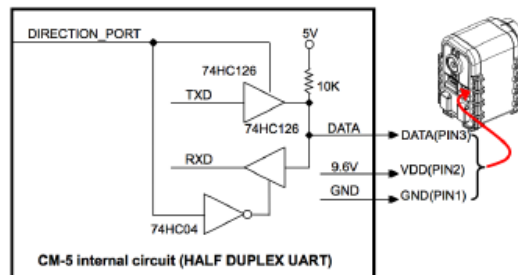


Imagen 8 Circuito Half Duplex UART

Fuente: Rutgers University. User's manual dynamixel Ax.12. Recuperado 15 de julio de 2016, de: <http://hackerspace.cs.rutgers.edu/library/Bioloid/doc/AX-12.pdf> . P7

Este circuito básicamente lo que hace es que cuando el motor este transmitiendo, el bus no esté conectado al pin Rx, y cuando este esté esperando recibir, que no esté siendo impulsada por el pin Tx.

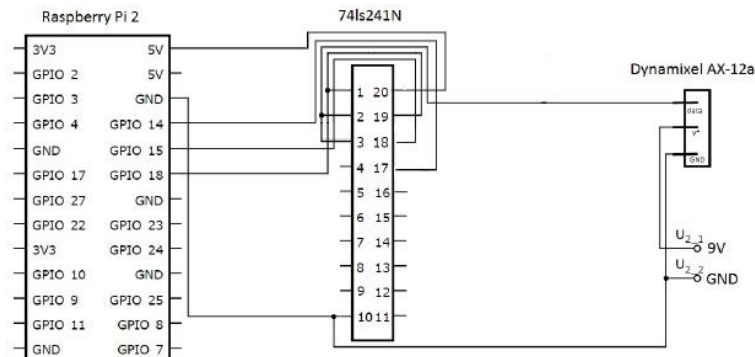


Imagen 9 Conexión motores con Raspberry pi

Fuente: Autor

Para que los motores quedaran de giro continuo se colocó que el ángulo fuera cero, de esta manera solamente se controlaba la velocidad del motor que se maneja de (0 a 1024) escrito en forma hexadecimal, para que se pueda hacer girar el motor hacia el otro sentido se maneja de (1024 a 2048) escrito en forma hexadecimal [12].

El funcionamiento de los motores se da por enviar un paquete de instrucciones el cual tiene la siguiente forma:

0xFF 0xFF ID LENGTH INSTRUCTION PARAMETER1 ... PARAMETER N CHECK SUM

Imagen 10 Paquete de instrucciones dynamixel ax-12A

Fuente: Rutgers University. User's manual dynamixel Ax.12. Recuperado 15 de julio de 2016, de: <http://hackerspace.cs.rutgers.edu/library/Bioloid/doc/AX-12.pdf> . P10

Donde los primero dos datos es la identificación de un paquete se está enviando, el tercer dato es el ID que es la identificación de cada motor que se esté usando, el cuarto dato es cuantas instrucciones se le van a dar al motor, las siguientes son las instrucciones o parámetros que se e van a enviar al motor como información adicional y el último dato es la suma de verificación [12] de los datos anteriores que tiene la siguiente forma:

$$\begin{aligned} check\ Sum = & \sim(ID + Length + Instruction \\ & + Parameter1 + .. Parameter N) \end{aligned} \quad (90)$$

Para el funcionamiento de los motores se realizaron mediante 2 códigos, el primero se muestra el funcionamiento de cómo se envía el paquete de instrucciones:

```
import serial
import time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

port = serial.Serial("/dev/ttyAMA0", baudrate=1000000,
timeout=0.001)
print "Program start"
time.sleep(3)
while True:

    GPIO.output(18, GPIO.HIGH)
```

```
port.write(bytearray.fromhex("FF FF 01 05 03 1E 32 03 A3"))
time.sleep(0.1)
GPIO.output(18, GPIO.LOW)
print "move to the left"
time.sleep(2)
```

```
GPIO.output(18,GPIO.HIGH)
port.write(bytearray.fromhex("FF FF 01 05 03 1E CD 00 0b"))
time.sleep(0.1)
GPIO.output(18,GPIO.LOW)
print "mov to the Right"
time.sleep(2)
```

Y el segundo código, es utilizando la librería que se usó para el movimiento de los motores:

```
from time import sleep
from lib2Motores import Ax12
from serial import Serial

object_Ax12 = Ax12()
object_Ax12.setAngleLimit(1,0x00,0x00)
object_Ax12.setAngleLimit(2,0x00,0x00)
```

while True:

```
    object_Ax12.moveSpeed(1,0x00,0x7e8)
    object_Ax12.moveSpeed(2,0x00,0x3e8)
    sleep(5)
    object_Ax12.moveSpeed(1,0x00,0x7e8)
    object_Ax12.moveSpeed(2,0x00,0x00)
    sleep(4)
```

La librería utilizada de los motores se encuentra en el anexo (H).

1.6 Atmega16

Atmega16 es un microcontrolador de alto rendimiento de la familia AVR de Atmel mega con bajo consumo de energía [13], se escogió este

microprocesador teniendo en cuentas las características principalmente la memoria, velocidad, distribución de pines y la comunicación por I2C.

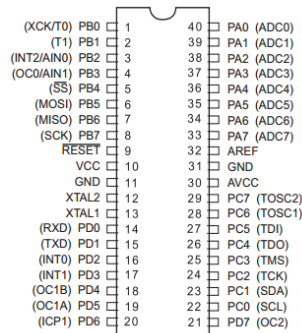


Imagen 11 Distribución de pines del microcontrolador ATmega16

Fuente: Atmel. 8-bit Microcontroller with 16K Bytes In-System Programmable Flash. Recuperado 15 de julio de 2016, de: <http://www.atmel.com/Images/2466S.pdf>

Características del microcontrolador ATmega16:

- CPU de 131 instrucciones.
- 32 registros de propósito general (cada uno de 8 bits).
- Memoria de programa flash de 16KBytes.
- Memoria SRAM de 1Kbyte.
- 32 líneas de entrada/salida.
- 3 timers (se puede generar hasta 4 PWM).
- ADC de 10 bits.
- Comparador Analógico.
- Comunicación serial USART, SPI, TWI, i2c.
- Timer Watchdog.
- Contador de tiempo real.

Este micro contralor cumple con leer los resultados de los sensores de ultrasonido y enviárselos a la Raspberry Pi mediante el protocolo de I2C, El Atmega16 fue programado sobre la plataforma de arduino IDE en el anexo (E) se muestra los pasos para poder programar el atmega16 con arduino IDE, se usó el atmega16 por el inconveniente de usar los sensores de ultrasonido con la Raspberry Pi, no mostraba valores esperados cuando se colocaban más de dos sensores ultrasonido.

1.7 Baterías

1.7.1 Batería Li-Po

La batería que se utilizó para los motores del robot móvil es una batería de polímero de litio (Li-Po), ya que este tipo de baterías por sus dimensiones, peso y su suministro de energía son ideales para el diseño del robot móvil, este tipo de baterías son recargables y capaces de liberar corrientes muy elevadas [14].

La batería seleccionada para el robot móvil es del fabricante ZIPPY COMPACT, con una tensión de salida de 11.1V y una corriente de 2200mah y cuenta con tres celdas, este tipo de baterías requieren un cuidado extra, ya que no se pueden dejar descargar por debajo de 3.0 V y necesitan de un cargador especial por tener tres celdas, esta batería cuenta con dos conectores, uno para la carga que es el conector JST-XH, y el otro para la descarga que es el XT60.



Imagen 12 Batería Li-Po

Fuente: HobbyKing. ZIPPY Compact 2200mAh 3S 25C Lipo Pack.
Recuperado 13 de septiembre de 2016 de:
http://www.hobbyking.com/hobbyking/store/__21346__ZIPPY_Compact_2200mAh_3S_25C_Lipo_Pack.html

1.7.2 Power Bank

Los Power Bank es una fuente de energía portátil constituido por varias baterías recargables, los Power Bank cuentan con una entrada para que se pueda de cargar y cuentan con uno o más puertos USB de salida para poder cargar o alimentar en nuestro caso la Raspberry Pi, el Power Bank

seleccionado fue uno que tuviera una salida de dos amperios, que indicara la carga tiene la batería y un tamaño y peso apropiados para el robot móvil.

Características del Power Bank.

- Capacidad 14000mah
- Voltaje de entrada 5V/1A
- Voltaje de Salida 5V/1A y 5V/2A
- Puerto de entrada Micro-USB
- Dimensiones 14.2x6.5x2.3cm



Imagen 13 Batería Power Bank

Fuente: Autor

1.8 Protocolo de comunicación I2c

I2c de su significado **Inter Integrated Circuit** es una aplicación de comunicación entre circuitos integrados, este bus define solo dos señales SDA(Serial DAta) y SCL(Serial CLock), el SDA define como una señal a drenado abierto ya sea este un dispositivo como esclavo o como maestro y SCL si se trata de un esclavo es una entrada y si es un maestro es una salida, el maestro genera una señal de sincronía entre todos los circuitos

integrados, este bus de datos necesita de resistencias que vayan a las líneas SDA y SCL y de estas a Vcc, para que puedan proporcionar un uno lógico ya que el excitador del i2c solo puede obtener el estado 0 lógico, de esta manera también se obtiene el aislamiento del bus colocando este nodo que se quiere aislar este en un estado alto[15].

Cada dispositivo que esté conectado a este bus de comunicación necesita una dirección para que pueda ser identificado, también debe operar como transmisor y receptor, la transmisión de datos es de 8-bits.

Los estados en el que opera este protocolo de comunicación es el siguiente:

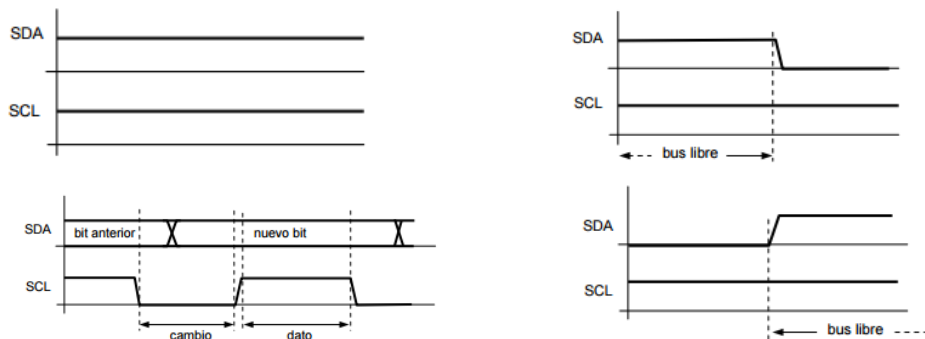


Imagen 14 Estados del Protocolo I2c

Fuente: Moreno A. The I2C Bus Specification. 2004. P186-187

En la imagen (a) es el estado libre el cual las dos líneas se encuentran en un estado alto sin ningún cambio, en la imagen (b) muestra un cambio a un estado bajo lo que indica que el bus está iniciando una transacción, la imagen (c) indica un cambio lo que quiere decir es que cuando la línea SCL está en bajo y en la línea SDA hay un cambio de estado, en este momento el sistema podrá poner los bits para transmitir, cuando el SCL vuelve a cambiar de estado en este caso es el dato y esto indica que este valor es válido para el sistema y la imagen (d) es la parada que ocurre cuando el SDA pasa de un estado bajo a un estado alto y se utiliza para poder indicar al esclavo que termino la transferencia y vuelve a quedar en estado libre el bus[15].

La conexión que se realizó entre los diferentes componentes que fueron necesarios para la construcción del robot móvil es la siguiente:

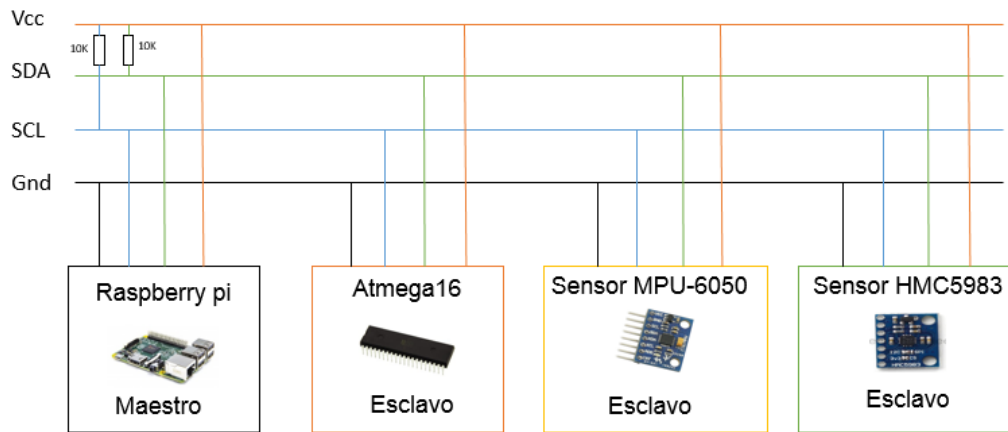


Diagrama 2 Conexión I2c

Fuente: Autor

1.9 Construcción del Robot móvil

La construcción del robot móvil se realizó en varios pasos el primero de ellos fue el diseño de la PCB, el cual se realizó en el software PCBWizard, se desarrollaron dos PCB, la primera para los motores Dynamixel y el integrado 74ls241 para que de esta manera se mantuvieran aisladas las dos fuentes que se utilizaron, el resultado de esta PCB es el siguiente:

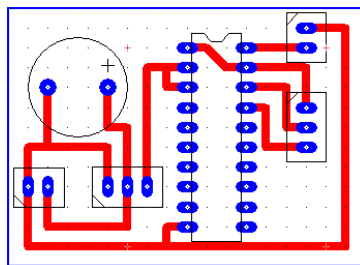


Imagen 15 Circuito motores Dynamixel e integrado 74ls241

Como se muestra en la imagen anterior se realizó la PCB con el fin de utilizar cables de fácil conexión ya que necesitamos conectar pines de la Raspberry Pi a las PCB.

La Segunda PCB consta de la mayoría de los sensores del agente robótico, como lo es la brújula, el acelerómetro, el atmega 16 y los sensores ultrasonido, la segunda PCB queda de la siguiente manera:

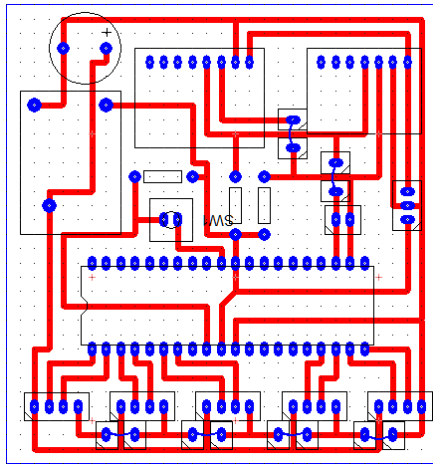


Imagen 16 PCB #2

Fuente: Autor

Ya al tener los circuitos hechos se diseñó la plataforma del robot móvil y se escogió como material el acrílico con medidas como se muestra en la siguiente imagen, como se puede observar la parte frontal de robot móvil se tiene esa estructura para que los sensores de ultrasonido puedan captar objetos desde 0° , 45° , 90° , 135° y 180° , esto con el fin de evitar cualquier obstáculo que se le presente.

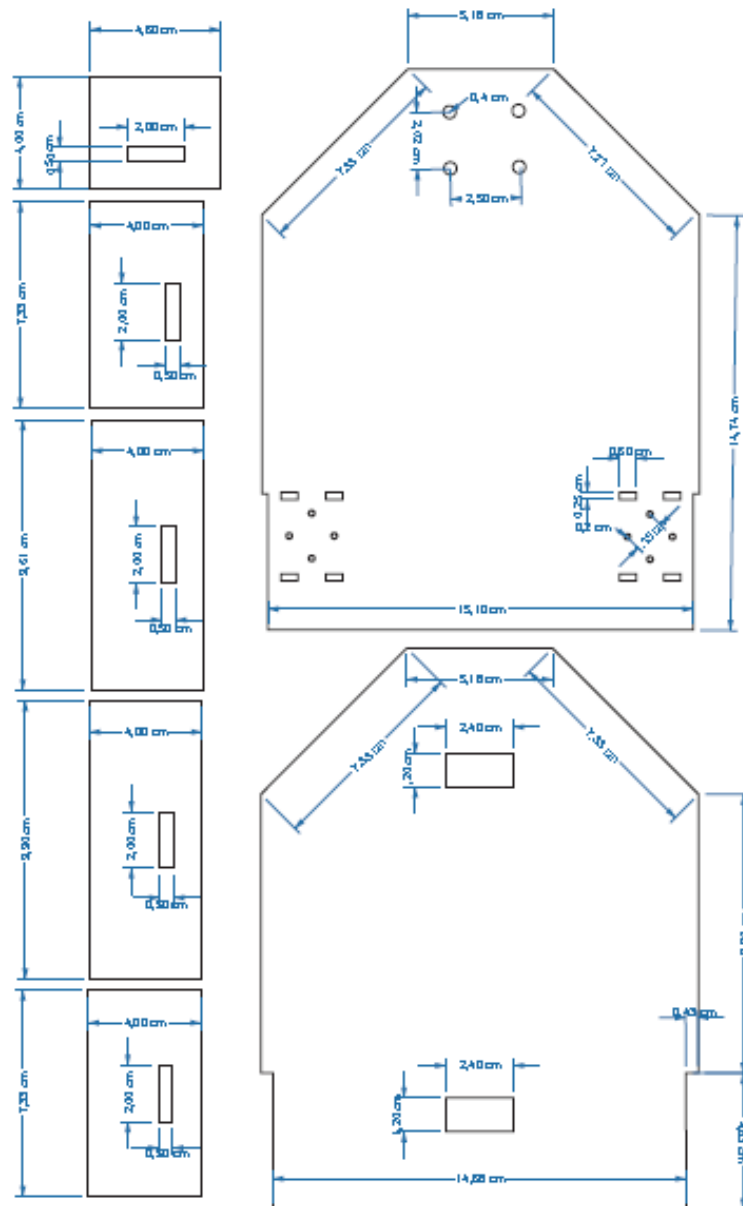


Imagen 17 Medidas Robot Móvil

Fuente: Autor

3. Procesamiento de imágenes

3.1 Odometría

La odometría es el estudio de la estimación de la posición de vehículos con ruedas duran su navegación, es importante que para la navegación de estos robot móviles se usa información como la rotación de las ruedas o también mediante secuencias de imágenes por una o varias cámaras, como se utiliza una cámara como en este caso se llama enfoque monocular o cuando se utilizan dos cámaras se llama enfoque estereoscópico, las etapas de la odometría son las siguientes.

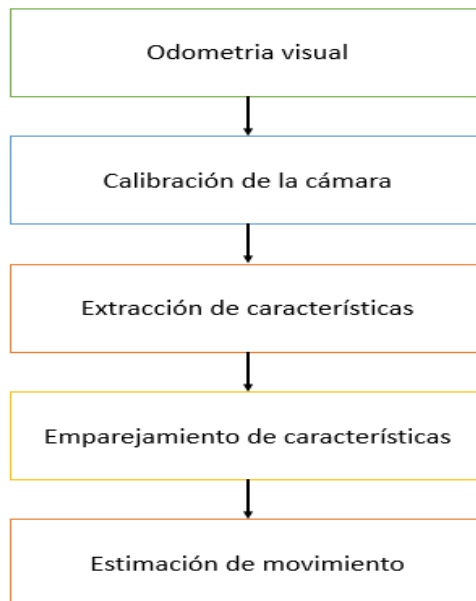
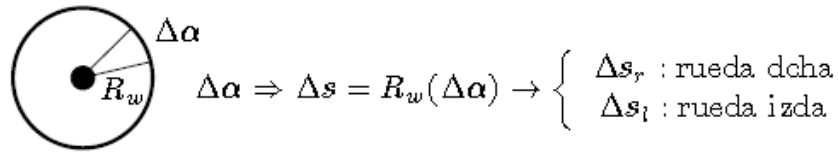


Diagrama 3 Etapas de la odometría

Fuente: Autor

La otra manera de poder determinar localizar el robot móvil es mediante la velocidad angular de las ruedas de tracción o la medición de giro de la rueda como se muestra en la siguiente imagen.



Fuente: Ortiz A. Navegación Para Robots Móviles

Los problemas de la odometria con una plataforma de tracción diferencial es que cada vez el error es más grande por la dificultad de generar caminos rectos por los problemas sistemáticos que presentan este tipo

Imagen 18 Giro de Rueda

de configuraciones.

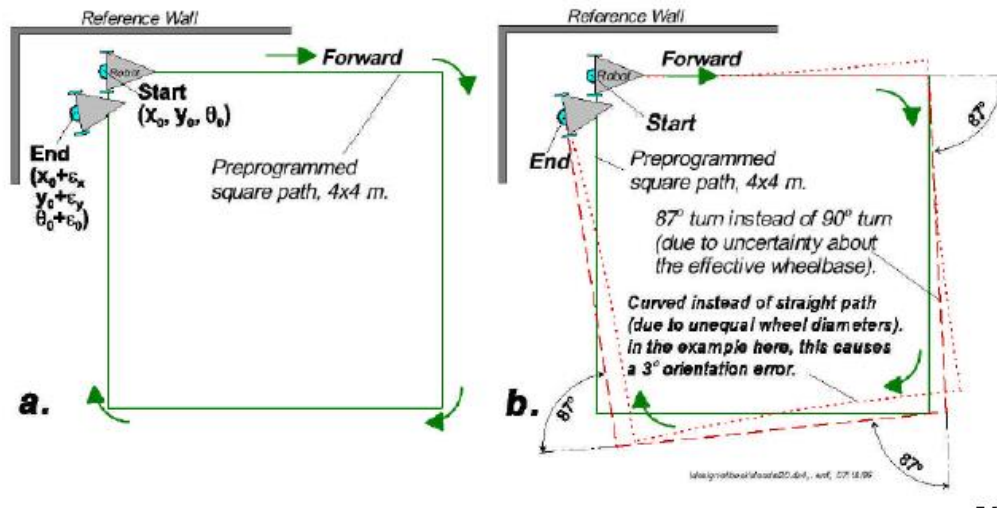


Imagen 19 Problemas de la Odometría

Fuente: Ortiz A. Navegación Para Robots Móviles

3.1.1 Calibración Cámara

Para la calibración de la cámara se utilizó la guía de calibración de cámara de opencv y Python, la cual trata el problema de la cámara como

dos tipos de distorsiones la radial y la tangencial, la distorsión radial trata el problema de que las líneas rectas se ven curvas en la cámara mediante la siguiente ecuación se resuelve.

$$X_{corregida} = x(1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (90)$$

$$Y_{corregida} = y(1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (91)$$

La distorsión tangencial se debe a que el lente de la cámara no está alineado y en este tipo de distorsión muestra problemas como algunas zonas de la imagen aparecen más cerca de lo que aparece y lo resuelven así.

$$X_{corregida} = x + [2p_1xy + p_2(r^2 + 2x^2)] \quad (92)$$

$$Y_{corregida} = y + [p_1(r^2 + 2x^2) + 2p_2xy] \quad (93)$$

Con las cuatro ecuaciones anteriores se debe encontrar 5 coeficientes:

$$\text{coeficientes de distorsion} = (k_1 \ k_2 \ p_1 \ p_2 \ k_3) \quad (94)$$

El primer paso que se realizó fue capturar varias imágenes con la cámara y tener un patrón definido, en este caso fue el siguiente:



Imagen 20 Patrón de calibración para la cámara

Fuente: Autor

El siguiente paso fue detectar la cuadrícula de la imagen y obtener los puntos de cada esquina del patrón de calibración para la cámara, como la imagen anterior se tomaron 7 imágenes pero con diferente ubicación y orientación, con el fin de poder determinar la matriz de la cámara que tiene parámetros extrínsecos e intrínsecos y se pueden expresar de forma matricial, primero mostrando el resultado que nos arroja el código que se muestra en el anexo (G).



Imagen 21 Detectar esquinas del patrón

Fuente: Autor

Ya detectados estos puntos sobre las 7 imágenes el programa arroja unos valores los cuales se puede crear la matriz de la cámara.

$$\text{Matriz de la camara} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & f_z \end{bmatrix} \quad (95)$$

Donde (f_x, f_y) es la longitud focal de la cámara y (c_x, c_y) son los centros ópticos.

$$\begin{aligned} &\text{Matriz de la camara} \\ &= \begin{bmatrix} 1.26207043e + 03 & 0.00000000e + 00 & 2.98248259e + 02 \\ 0.00000000e + 00 & 1.58875842e + 03 & 3.42531319e + 02 \\ 0.00000000e + 00 & 0.00000000e + 00 & 1.00000000e + 00 \end{bmatrix} \end{aligned} \quad (96)$$

Obteniendo la nueva matriz de la cámara, el resultado comparando la imagen sin calibrar y después de calibrar es la siguiente.

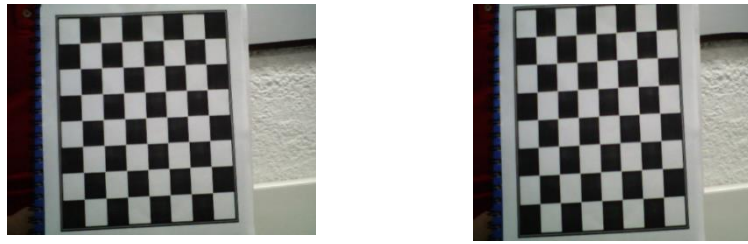


Imagen 22 Resultado de la Calibración

Fuente: Autor

3.2 Land-marks:

Un Land-marks es un elemento que se encuentra en el espacio donde está el robot, los Land-Marks se caracterizan por resaltar sobre los demás objetos en el entorno, existen algoritmos para poder detectar características propias del entorno y crear los Land-marks a partir del entorno en donde se desenvuelve el robot como esquinas en un espacio cerrado o riscos en un espacio abierto, en este caso lo que se realizó fue crear los Land-marks con imágenes propias las cuales el robot debe reconocer y asignarles una identificación única. Con la creación de los Land-Marks ayuda al robot a poder localizarse y poder tomar decisiones y reconocer si ya pasó por este Land-marks o es la primera vez que ve el Land-Mark.



Imagen 23 Land-Marks

Fuente: Ortiz A. Navegación Para Robots Móviles

3.3 Identificación de los Land-Marks

La identificación de los Land-Marks se realizó por medio de rectángulos con colores que usualmente no se encuentren en entornos cerrados, ubicados en diferentes partes del espacio para la localización del robot, el centro del rectángulo es la información que se necesita obtener para poder determinar la acción que debe tomar.

El primer paso que se realizó fue a la asignación del color y realizar pruebas en el programa OPENCV, La instalación de OPENCV sobre la Raspberry pi se encuentra en el anexo (6).

3.3.1 Modelo de colores HSV

OPENCV trabaja bajo el modelo de colores HSV ya que este modelo permite seleccionar un color en específico, la función de OPENCV toma los valores en RGB y los pasa a HSV, el cono de colores HSV:

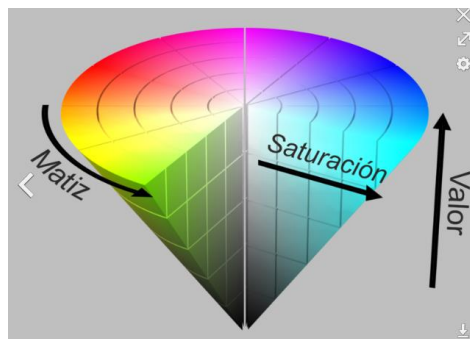


Imagen 24 Cono Colores HSV.

Este cono de colores el matiz tiene valores de 0° a 360° en el que cada valor corresponde a un color por ejemplo el rojo es 0, verde es 120, la saturación indica el brillo entre el negro-blanco y el valor sobre el eje vertical blanco y negro, donde su valor sea máximo sería un color blanco.

El primer ejemplo que se realizó en OPENCV con los siguientes resultados.

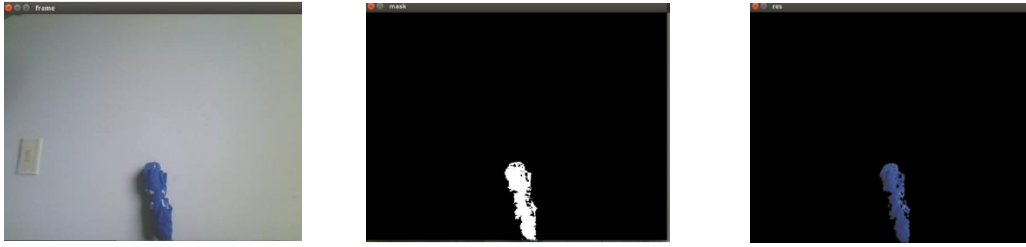


Imagen 25 Ejemplo OPENCV

Fuente: Autor

El código se encuentra en el anexo (7)

Para la calibración del color se utilizó un programa el cual permite tener un rango modificando los valores HSV y de esta manera filtrar el color en específico que va a tener el Land-Mark creando una máscara para determinar que la cámara solo detecte ese color entre los demás colores en el entorno que se mueva el robot móvil.

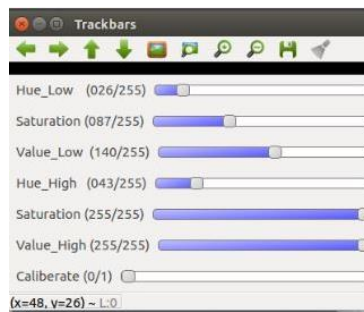


Imagen 26 Tabla calibración del color HSV

Fuente: Autor

Con esta tabla se pudo ajustar el color que se deseó en el entorno con el siguiente resultado.



Imagen 27 Color Utilizado para el Land-mark y la mascara

Fuente: Autor

Como se muestra en las imágenes anteriores se pudo filtrar el color, para posteriormente obtener la información que está contenida por el rectángulo, ya que esta es la información que necesita saber el robot móvil para poder tomar una decisión y de esta manera poder ubicarse en el entorno, el proceso que se le realizó o las técnicas de segmentación en la imagen para poder identificarlas.

3.3.2 Técnicas de segmentación

Las siguientes operaciones morfológicas simplifican las imágenes y se rescata o se salva la información principal de las imágenes, la morfología tiene como objetivo la supresión de ruido, destacar la estructura de los objetos y descripción de los objetos ya sea su área o perímetro.

3.3.2.1 Binarización de imágenes

La binarización de una imagen consiste en que los únicos valores posibles para un pixel sean 0 o 1 que para la representación gráfica será un blanco o un negro, con este paso se reduce la información de la imagen, colocando un umbral el cual dependiendo del valor del pixel se decida si es 0 o 1.

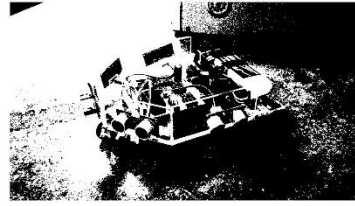
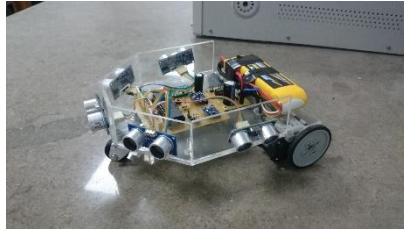


Imagen 28 Binarizacion de una imagen

Fuente: Autor

3.3.2.2 Erosión de imágenes

La erosión en una imagen elimina los objetos de poco importancia o muy simples, es decir que los objetos muy complicados pueden quedar descompuestos en objetos más simples de interpretar.

$$A \ominus B = \{x | B_x \subseteq A\} \quad (97)$$

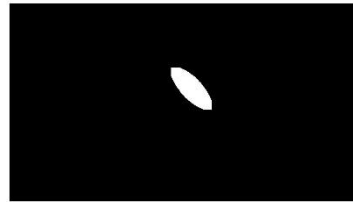
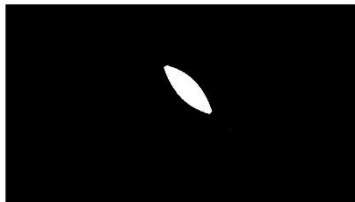


Imagen 29 Erosión

Fuente: Autor

3.3.2.3 Dilatación de imágenes

La dilatación de una Imagen es resaltar o ampliar el tamaño de los objetos que se encuentran en la imagen con el fin o el proceso más utiliza es erosionar y luego dilatar para quitar el ruido de la imagen y poder dejar la imagen con la información que se necesita.

$$A \oplus B = \{x | (B_x) \cap A \neq \emptyset\} \quad (98)$$



Imagen 30 Dilatación

Fuente: Autor

3.3.2.4 Cambio de tamaño de una imagen.

Para cambiar de tamaño en una imagen se utilizan dos conceptos, interpolación y diezmado, la interpolación es aumentar el número de píxeles de una parte de la imagen pero esto quiere decir que la imagen pierde nitidez.



Imagen 31 Interpolación

El diezmado es lo contrario quitar píxel a la imagen.

3.3.3 Detección del Land-Mark

Para la detección del land-Mark utilizando el programa anterior se realizó el siguiente algoritmo.

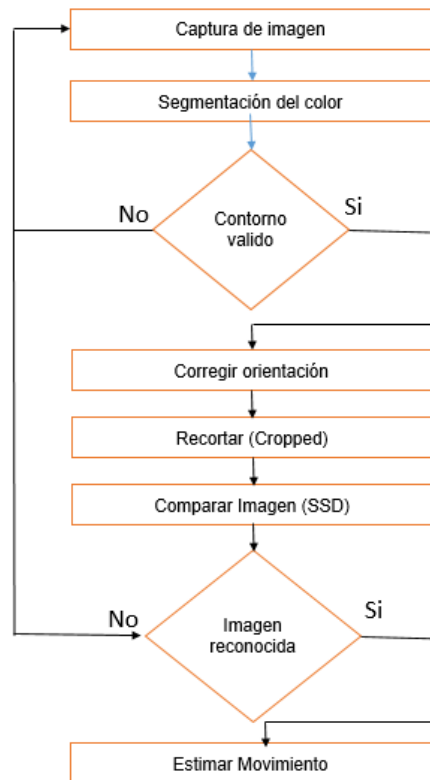


Diagrama 4 Algoritmo detectar el Land-Mark

Fuente: Autor

El banco de Land-marks utilizados son los siguientes.

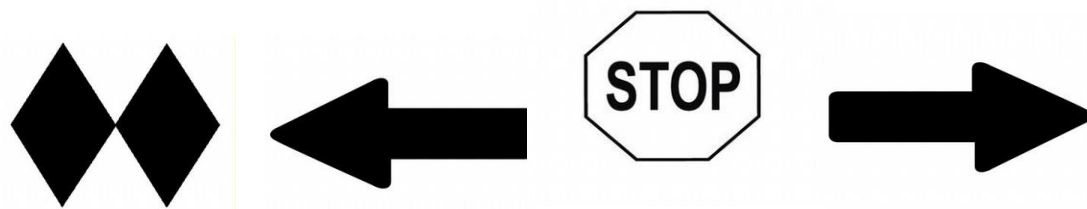


Imagen 32 Land-Marks

Fuente: Autor

El primer paso se explicó en la sección anterior de cómo obtener el filtrado del color que necesitamos, para el segundo paso se utilizó una función de OPENCV la cual se le dio un área mínima para que encuentre el contorno y se dibujó un rectángulo hallando los cuatro puntos del contorno y se muestra el siguiente resultado.

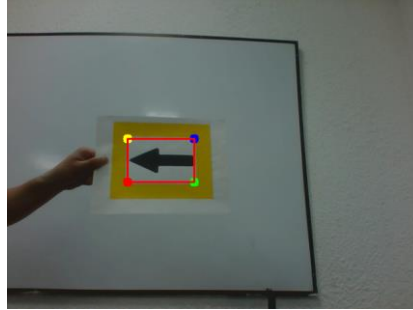


Imagen 33 Rectángulo del contorno del Land-Mark

Fuente: Autor

Lo que se realizó después de obtener el rectángulo fue corregir cuando el Land-Mark este girado, se realizó la función mediante el siguiente método:

Primero Para determinar el ángulo de la imagen se utiliza la matriz de rotación la cual tiene la siguiente:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad (8)$$

La matriz esta girada con respecto al centro de la imagen la cual nos arroja un ángulo mediante la siguiente función:

giro = cv2.fitEllipse(cnt)
angle = (giro[2] - 90) % 360

Este ángulo se le resta los 90 grados para poder dejar el cero de forma horizontal a la captura de la imagen, para encontrar el ángulo nuevo mediante la siguiente ecuación.

$$\theta_2 = -\arctan\left(\frac{y}{x}\right) + angle \quad (99)$$

Ya que se obtuvo el ángulo en el que se encuentran los puntos del rectángulo lo siguiente es obtener la magnitud que hay del centro de la imagen a cada uno de los puntos para esto se realizó mediante la siguiente ecuación.

$$r = \sqrt{x^2 + y^2} \quad (100)$$

Ahora para obtener los nuevos puntos del rectángulo ya girado mediante la siguiente ecuación:

$$\sin \theta = \frac{y}{r} \quad (101)$$

$$\cos \theta = \frac{x}{r} \quad (102)$$

Despejando x y y , se obtienen los nuevos puntos solo falta ajustarle otra vez la distancia que tienen con respecto al centro de la imagen, entonces la ecuación queda de la siguiente manera.

$$x' = \text{centro}[0] + r * \cos\theta \quad (103)$$

$$y' = \text{centro}[1] + r * (-\sin\theta) \quad (104)$$

De esta manera se giró el rectángulo para poder seguir con el siguiente paso para la detección del land-Mark.

El algoritmo implemento ya en Python quedo de la siguiente manera.

def box_correction(box,center,angle)

box2 = [[],[],[],[]]

for i in range(0, 4):

x = box[i][0] - center[0]

y = box[i][1] - center[1]

angle2 = (math.degrees(-math.atan2(y, x)) + angle) % 360

r = math.sqrt(math.pow(x, 2) + math.pow(y, 2))

```

posX = center[0] + r*math.cos(math.radians(angle2))
posY = center[1] + r*math.sin(math.radians(-angle2))

```

```

box2[i].append(int(posX))
box2[i].append(int(posY))

```

```

box2 = np.array(box2)
return box2

```

El resultado del anterior código es el siguiente:

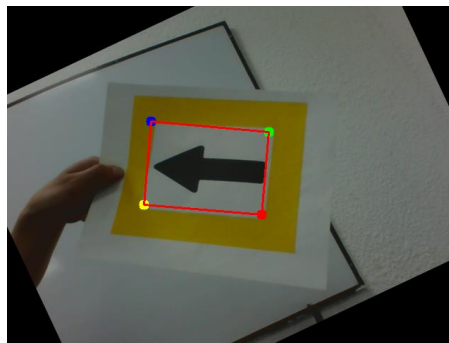


Imagen 34 Land-Mark girado

Fuente: Autor

La corrección anterior se realizó ya que si la imagen esta girada al realizar el error cuadrático medio no iba a coincidir, entonces toca dejar la imagen siempre horizontalmente por eso se realizó la función anterior.

Al obtener el cropped o la sección interna del rectángulo lo primero que se realizo fue dejar este recorte con el mismo tamaño de la imagen que esta guardada en el banco de imágenes, al tener las dos imágenes del mismo tamaño se binarizo la imagen y luego se erosiono y dilato con la misma mascara para poder quitar el ruido de la imagen, en las siguientes imágenes se muestra el proceso que se le realizo al cropped.



(a)



(b)

(c)

Imagen 35 Proceso Cropped

Fuente: Autor

En las imágenes anteriores se muestra el proceso realizado al cropped de la imagen original, en la imagen (a) es el cropped original sin ningún proceso de segmentación, en la imagen (b) es el cropped con el proceso llamado *resize* es decir se ajustó el cropped con el mismo tamaño de la imágenes del banco de pruebas, en la imagen (c) es el resultado del proceso de binarización, dilatación y erosión que se le realizó a la imagen para que pueda ser comparada con las imágenes del banco de pruebas.

El método que se utilizó para la comparación de las imágenes fue mediante el error cuadrático medio (SSD), el cual es un indicador que permite medir el error entre dos imágenes que tengan el mismo tamaño, la ecuación del error cuadrático medio es la siguiente:

$$SSD = \sqrt{\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (\bar{f}(x,y) - f(x,y))^2} \quad (105)$$

Donde $f(x,y)$ es la imagen original y $\bar{f}(x,y)$ es la imagen experimental en este caso sería nuestro cropped, en esta formula si el resultado es cero quiere decir que las dos imágenes son iguales, la implementación de esta ecuación se muestra en el siguiente código:

```
def ssd(im_ref,crop):
    ssd_1 = []
    ssd = 0
    for i in range(0,4):

        a = im_ref[i]
        o2 = resize_y_binario(a)

        ssd = 0
        for M in range(h/2):
            for N in range(w/2):
```

```

        ssd += abs((1/M*N)*int(crop[M,N])-int(o2[M,N]))
    ssd_1.append(ssd)
ssd_1 = np.array(ssd_1)
return ssd_1

```

Lo que retorna la anterior función son los errores que tiene el cropped con las cuatro imágenes que se encuentran en el banco de imágenes el código completo se encuentra en el anexo (J), los resultados que se obtuvo para poder determinar el rango de errores de las siguientes imágenes son los siguientes.

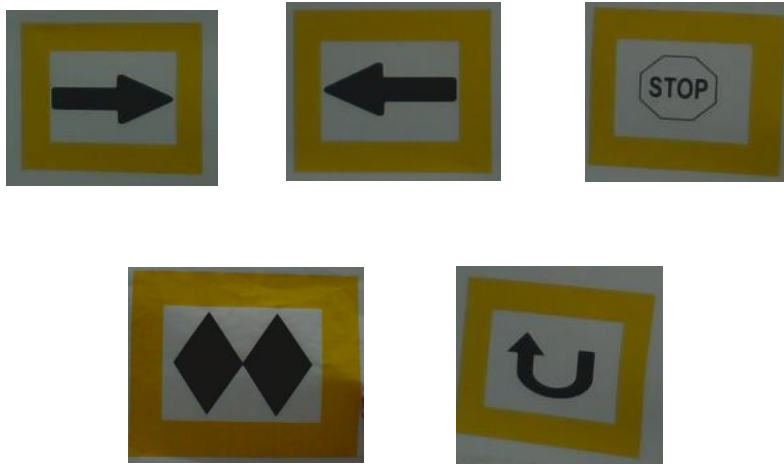


Imagen 36 Comparación de errores de los Land-Marks

Tabla 2 Promedio de errores entre los Land-Marks



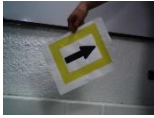

	Giro_D	Giro_I	Go	Stop	Otra_I
Giro_D	9981	20300	39165	35823	40910
Giro_I	20871	9423	39529	39739	39020
Go	37551	32699	10262	57229	44717
Stop	36795	39232	57353	11070	27625




Fuente: Autor




La tabla anterior es un promedio de los errores que tienen las imágenes con respecto al cropped, se pudo observar que el error nunca va hacer cero ya que por más que se implementen las técnicas de segmentación la imagen tiene ruido, pero de todas maneras se nota un error promedio de todas la imágenes menor a (12000), a partir de este valor se pueden detectar las imágenes.

Se realizó un prueba para poder mejorar el tiempo de computo del programa y volverlo más eficiente, la prueba que se hizo fue comparar los resultados de las imágenes con la calibración y sin la calibración ya que esta función consume tiempo de máquina, se ubicaron los land-Marks en diferentes posiciones con respecto a la cámara para poder determinar el comportamiento de los datos.

Tabla 3 Comparación error de la Calibración

Imagen	Error sin Calibración	Error Con Calibración
	4069	5789
	3279	4516
	4119	3950
	No la reconoció	No la reconoció

	3317	2975
	No la reconoció	No la reconoció
	3709	No la reconoció

	No la reconoció	No la reconoció
	3599	3691
	2168	2356

Fuente: Autor

Con los resultados obtenidos de la tabla anterior se pudo determinar que cuando se realiza la calibración a la imagen el error (SSD) aumenta en la mayoría de la imágenes y se alcanza a perder una imagen de 10 capturas y por esta razón se decidió no realizarle la calibración a la cámara y por qué el tiempo de maquina aumenta.

4. Estimación de Movimiento

En este capítulo se presenta como se determinó el desplazamiento del robot móvil en el entorno; el primer problema que se tuvo fue determinar la velocidad angular de las ruedas traseras para poder estimar el recorrido del robot ya que la odometria se usa mediante encoders pero en este caso los motores dynamixel no poseen enconders internos, entonces lo que se planteo fue que aprovechando el control PID que poseen los motores dynamixel, se realizó una serie de pruebas para estimar la velocidad apropiada para el robot.

Para indicarle al robot móvil que velocidad debe ir se utiliza mediante la siguiente línea:

object_Ax12.moveSpeed(1,0x00,0x250)

object_Ax12.moveSpeed(2,0x00,0x250)

Donde el primer valor es el ID de cada motor y el tercer valor es un numero en hexadecimal que indica a qué velocidad va a estar el motor,

mediante esto se establecieron velocidades fijas y se realizaron pruebas para determinar la velocidad lineal y su velocidad angular, en este parte un motor tiene un valor de velocidad de (0 a 1024) y el otro de (1024 a 2048), se establecieron los siguientes valores para cada motor.

Tabla 4 Valores asignados Ruedas Traseras

Rueda ID 1	Rueda ID 2
1324 -- 0x52c	304 -- 0x130
1279 -- 0x4ff	259 -- 0x103
1174 -- 0x496	154 -- 0x9a
1099 -- 0x44b	79 -- 0x4f

Fuente: Autor

Con estos valores ya fijos se realizaron pruebas para determinar la velocidad lineal y velocidad angular de cada rueda mediante las siguientes formulas:

$$V = \frac{d[cm]}{t[s]} \quad (9)$$

$$W = \theta[rad/s] * r[cm] \quad (10)$$

Radio de las ruedas = 2.6cm

Tiempo = 5s

Primera Prueba:

Rueda 1 = 0x52c

Rueda 2 = 0x130

Tabla 5 Velocidad del robot móvil prueba 1

Robot Móvil	
Distancia(cm)	Velocidad Lineal(d/t)
25,6	5,12
25,7	5,14
25,7	5,14
25,7	5,14
25,8	5,16

Fuente: Auto

Tabla 6 Velocidad Rueda 1 = 0x52c

Rueda 1			
grados	Velocidad Angular (°/s)	Velocidad Angular (rad/s)	Velocidad lineal (velang*rad)
575	115	2,00712864	5,218534463
565	113	1,972222055	5,127777342
570	114	1,989675347	5,173155903
570	114	1,989675347	5,173155903
570	114	1,989675347	5,173155903

Fuente: Autor

Tabla 7 Velocidad Rueda 2 = 0x130

Rueda 2			
grados	Velocidad Angular (°/s)	Velocidad Angular (rad/s)	Velocidad lineal (velang*rad)
570	114	1,989675347	5,173155903
575	115	2,00712864	5,218534463
575	115	2,00712864	5,218534463
570	114	1,989675347	5,173155903
575	115	2,00712864	5,218534463

Fuente: Autor

Segunda Prueba:

Rueda 1 = 0x4ff

Rueda 2 = 0x103

Tabla 8 Velocidad del Robot Móvil prueba 2

Robot Móvil	
Distancia(cm)	Velocidad Lineal(d/t)
18	3,6
18,4	3,68
18	3,6
18,1	3,62
18,4	3,68

Fuente: Autor

Tabla 9 Velocidad Rueda 1 = 0x4ff

Rueda 1			
grados	Velocidad Angular (°/s)	Velocidad Angular (rad/s)	Velocidad lineal (velang*rad)
402	80,4	1,403244719	3,648436268
410	82	1,431169987	3,721041965
402	80,4	1,403244719	3,648436268
400	80	1,396263402	3,630284844
405	81	1,413716694	3,675663405

Fuente: Auto

Tabla 10 Velocidad Rueda 2 = 0x103

Rueda 2			
grados	Velocidad Angular (°/s)	Velocidad Angular (rad/s)	Velocidad lineal (velang*rad)
410	82	1,431169987	3,721041965
405	81	1,413716694	3,675663405
415	83	1,448623279	3,766420526
415	83	1,448623279	3,766420526
410	82	1,431169987	3,721041965

Fuente: Autor

Tercera Prueba:

Rueda 1 = 0x496

Rueda 2 = 0x9a

Tabla 11 Velocidad Robot Movil prueba 3

Robot Móvil	
Distancia(cm)	Velocidad Lineal(d/t)
11,2	2,24
11	2,2
11,4	2,28
11,2	2,24
11,5	2,3

Fuente: Autor

Tabla 12 Velocidad Rueda 1 = 0x496

Rueda 1			
grados	Velocidad Angular (°/s)	Velocidad Angular (rad/s)	Velocidad lineal (velang*rad)
250	50	0,872664626	2,268928028
240	48	0,837758041	2,178170906
250	50	0,872664626	2,268928028
255	51	0,890117919	2,314306588
265	53	0,925024504	2,405063709

Fuente: Autor

Tabla 13 Velocidad Rueda 2 = 0x9a

Rueda 2			
grados	Velocidad Angular (°/s)	Velocidad Angular (rad/s)	Velocidad lineal (velang*rad)
255	51	0,890117919	2,314306588
250	50	0,872664626	2,268928028
260	52	0,907571211	2,359685149
260	52	0,907571211	2,359685149
250	50	0,872664626	2,268928028

Fuente: Autor

Cuarta Prueba:

Rueda 1 = 0x44b

Rueda 2 = 0x4f

Tabla 14 Velocidad Robot Móvil Prueba 4

Robot Móvil	
Distancia(cm)	Velocidad Lineal(d/t)
5	1
5,2	1,04
5	1
5,3	1,06
5,2	1,04

Fuente: Autor

Tabla 15 Velocidad Rueda 1 = 0x44b

Rueda 1			
grados	Velocidad Angular (°/s)	Velocidad Angular (rad/s)	Velocidad lineal (velang*rad)
100	20	0,34906585	0,907571211
105	21	0,366519143	0,952949772
100	20	0,34906585	0,907571211
100	20	0,34906585	0,907571211
95	19	0,331612558	0,86219265

Fuente: Autor

Tabla 16 Velocidad Rueda2 = 4f

Rueda 2			
grados	Velocidad Angular (°/s)	Velocidad Angular (rad/s)	Velocidad lineal (velang*rad)
95	19	0,331612558	0,86219265
100	20	0,34906585	0,907571211
100	20	0,34906585	0,907571211
105	21	0,366519143	0,952949772
100	20	0,34906585	0,907571211

Fuente: Autor

En resumen de las tablas anteriores se puede determinar que los valores asignados a cada rueda representan una velocidad angular y lineal promedio como se muestra en la siguiente tabla:

Tabla 17 Velocidades determinadas

Rueda ID 1	Rueda ID 2	VEL (cm/s)	VEL-ANG RUEDA 1	VEL-ANG RUEDA 2
1324 -- 0x52c	304 -- 0x130	5,1	1,985	1,988
1279 -- 0x4ff	259 -- 0x103	3,63	1,409	1,43
1174 -- 0x496	154 -- 0x9a	2,25	0,879	0,89
1099 -- 0x44b	79 -- 0x4f	1,028	0,349	0,349

Fuente: Autor

Ya con unas velocidades determinadas el siguiente paso fue hacer uso de una herramienta de Python que se llama hilos, esta definición se puede entender como ejecutar varias tareas al mismo tiempo sin tener que parar o detener la ejecución del programa principal, el uso que se le dio a esta cualidad fue crear un contador para poder estimar la distancia recorrida por el robot móvil con las diferentes velocidades, de esta forma se construyó un enconder con unas velocidades predeterminadas y con un contador el cual no va a interrumpir el programa principal, para poder estimar la distancia recorrida.

5. Mapas

El tipo de mapa que se quiso representar es un mapa topológico el cual se basa en las relaciones geométricas entre características del en lugar de su posición absoluta, este tipo de mapas se representa por medio de grafos que son los nodos los cuales pueden indicar características especiales del entorno o también llevan asociado información geométrica como su posición respecto algún sistema de coordenadas [16].

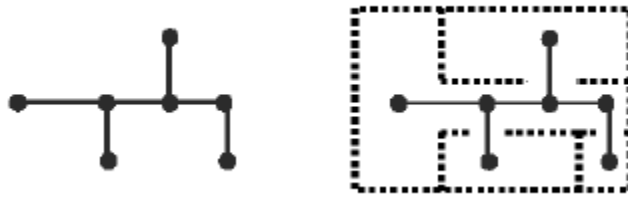


Imagen 37 Mapa Topológico

Fuente: Modelado del entorno en robótica Móvil

Para la representación del recorrido hecho por el robot móvil, se basó sobre la ecuación de la cinemática directa la cual nos da las velocidades que tiene el robot móvil en un instante de tiempo.

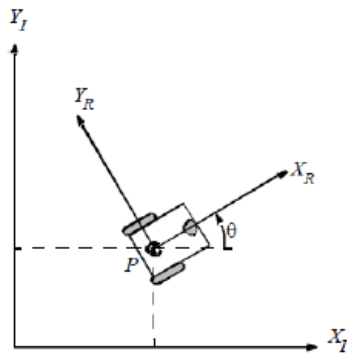


Imagen 38 ubicación del robot móvil

Fuente: Modelado del entorno en robótica Móvil

$$\dot{x} = \frac{(\theta_i + \theta_d)r}{2} \sin \Psi$$

$$\dot{y} = \frac{(\theta_i + \theta_d)r}{2} \cos \Psi$$

$$\dot{\Psi} = \frac{(\theta_i - \theta_d)r}{l}$$

Mediante las ecuaciones anteriores se puede saber la trayectoria del agente robótico tomando como Ψ los datos que arroja el sensor HMC5983, la brújula digital estos datos los tomamos en radianes, y luego como \dot{x} y \dot{y} son velocidades en cada uno de los ejes lo que se realizó fue integrar estas velocidades y como en el capítulo anterior se estableció unas velocidades angulares, podemos conocer la posición del robot en un instante de tiempo dado.

Las siguientes pruebas que se determinaron fueron para comprobar si la toma de datos y la gráfica era adecuada para el cumplimiento con los requisitos establecidos

Prueba 1:

Recorrido del robot móvil con la misma velocidad angular para las dos ruedas, iniciando en un ángulo $\frac{3\pi}{4}$ el resultado se muestra en la siguiente imagen

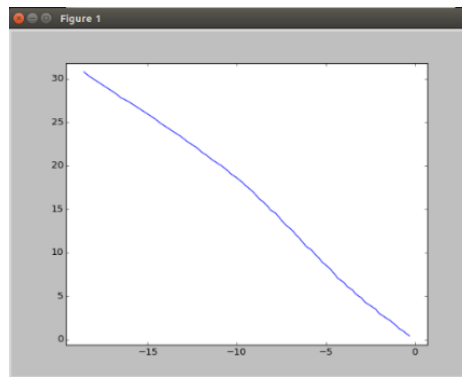


Imagen 39 Recorrido con la mismas velocidad angular en cada rueda

Fuente: Autor

Prueba 2:

Recorrido del robot con un velocidad angular en la rueda izquierda de $0.34 \text{ [[rad][cm/s]]}$ y una velocidad angular en la rueda derecha de $1.9 \text{ [[rad][cm/s]]}$

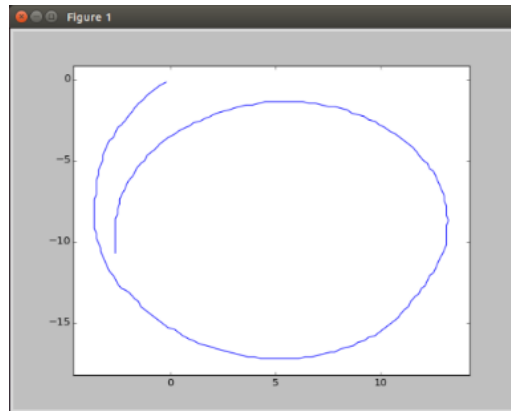


Imagen 40 Recorrido del robot con diferentes velocidades angulares

Fuente: Autor

Prueba 3

El siguiente recorrido se determinó que después de una distancia recorrida, gire dos veces a la izquierda y luego gire a la derecha, de esta manera como se ve en la siguiente imagen se puede observar los resultados del recorrido que se deseó realizar.

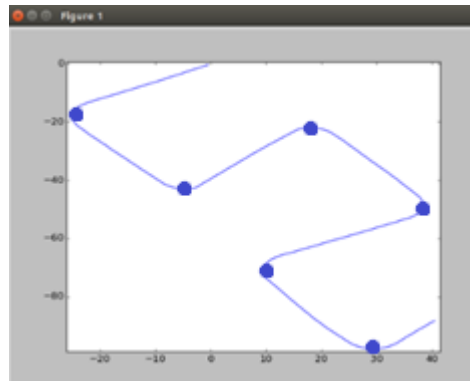


Imagen 41 Recorrido del robot, trayectoria predefinida

Fuente: Autor.

Luego de la estimación del recorrido, se elaboró un pequeño mapa en el cual el robot se pueda desplazarse evitando obstáculos y reconociendo los Land-Marks, el entorno en el que se ejecutaron la pruebas se muestra en la siguiente imagen.

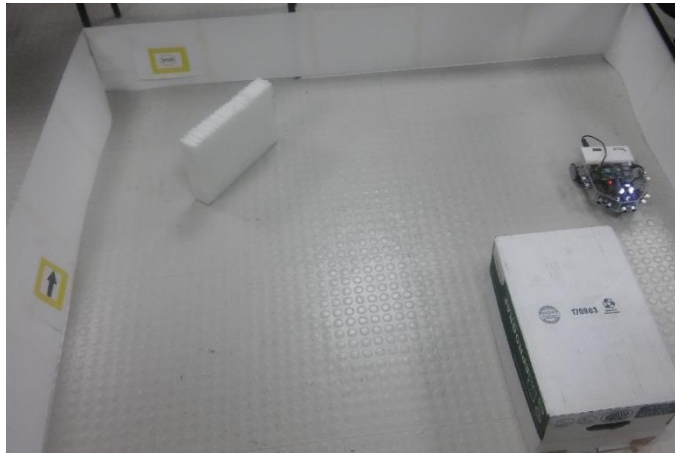


Imagen 42 Escenario para las pruebas del robot

Fuente: Autor

Sobre este escenario se realizaron las pruebas, dejando recorrer el robot móvil de diferentes posiciones para obtener las rutas en las que el agente robótico se desplazó.

En la primera prueba que se realizó el robot móvil comenzó como muestra la siguiente imagen:

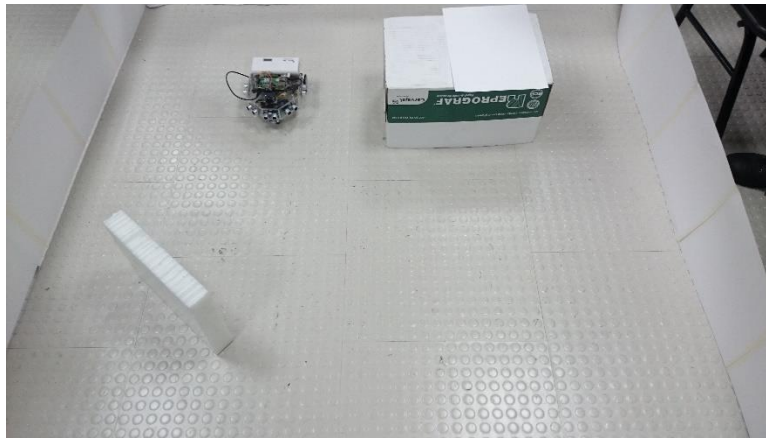


Imagen 43 Primera Prueba de Mapeo

Fuente: Autor.

El resultado de esta primera prueba del recorrido del robot sobre este escenario se muestra en la siguiente imagen.

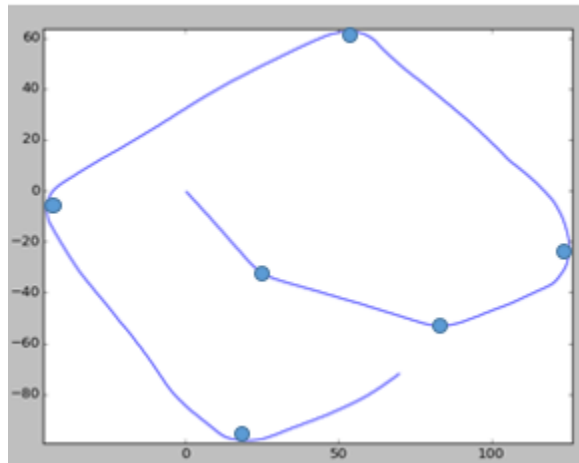


Imagen 44 Resultado primera Prueba

Fuente: Autor.

Como se muestra en la imagen anterior el robot móvil genera una trayectoria comenzando desde una orientación de $\frac{5\pi}{3}$ y generando un recorrido por donde es posible que se pueda navegar en el mapa, en el segundo experimento se ubica la plataforma móvil en otra posición con diferente orientación para poder estimar los obstáculos que se encuentran en el mapa.

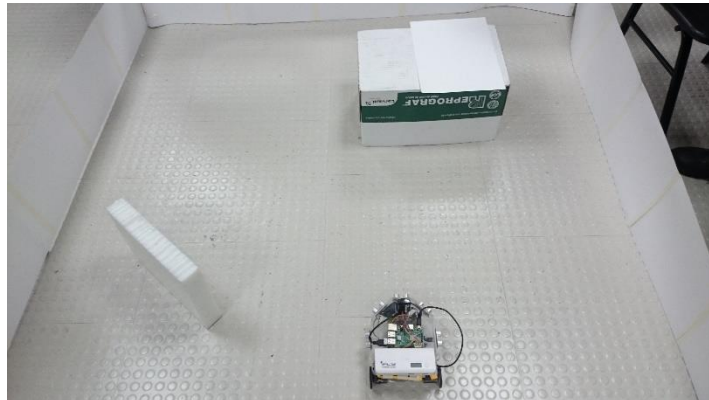


Imagen 45 Segunda prueba de Mapeo

Fuente: Autor.

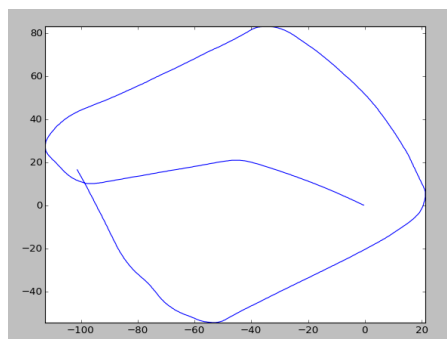


Imagen 46 Resultado Segunda prueba

Fuente: Autor

CONCLUSIONES

Después de realizar una revisión de los métodos de segmentación y comparación de imágenes en los cuales se basa la odometría se concluye que los métodos más eficientes para este objetivo son el error cuadrático medio, y la correlación transversal normal. Después de realizar un análisis comparativo se determina que el método óptimo a implementar es el error cuadrático debido a que requiere menos recursos computacionales.

Mediante el uso de las ecuaciones cinemáticas se logró cumplir con el objetivo de representar gráficamente el desplazamiento del robot, obteniendo a través de dichas gráficas, el mapa de la trayectoria seguida por el vehículo durante su operación.

En la actualidad existen plataformas móviles enfocadas al mapeo y localización que utilizan altos recursos computacionales, los cuales son capaces de generar múltiples procesos al mismo tiempo; lo que lleva a que su costo de construcción sea muy elevado debido a que se necesitan equipos informáticos de alto nivel; con el fin de reducir costos en la construcción de la plataforma móvil, se implementaron nuevas técnicas o algoritmos, los cuales integran múltiples sensores de bajo costo, eficientes desde el punto de vista computacional, que a su vez permiten obtener resultados similares a las plataformas existentes en el mercado.

RECOMENDACIONES

Se recomienda operar la plataforma móvil en superficies planas y lisas para obtener un mejor comportamiento del vehículo y de esta forma reducir errores en la trayectoria del mismo, que se puedan presentar debido a irregularidades en el terreno.

Implementar sensores de ultra sonido de mayor precisión a los usados en el vehículo desarrollado, que permitan mayor exactitud en el movimiento de la plataforma móvil.

Con fin de optimizar los procesos que conllevan la caracterización del vehículo, se recomienda reemplazar los motores actualmente utilizados, por motores que incorporen encoders, debido a que con ellos, se obtiene instantáneamente la velocidad angular que tiene el motor en movimiento.

Se recomienda incorporar al vehículo, una cámara de mejores características que permita obtener imágenes de mejor calidad a las que actualmente capta la cámara usada.

Se recomienda optimizar la plataforma gráfica para poder generar un mapeo más detallado que el que actualmente es generado.

REFERENCIAS

- [1] Viñals. Localización y generación de mapas del entorno (SLAM) de un robot por medio de una Kinect.2012
- [2] Bailey,Durrant . Simultaneous Localization and Mapping (SLAM): Part II.2006
- [3] Auat Cheeín, Sciascio, Carelli .Planificación de Caminos y Navegación de un Robot Móvil en Entornos Gaussianos mientras realiza tareas de SLAM.2008
- [4] Fuentes. LOCALIZACIÓN Y MODELADO SIMULTÁNEOS EN ROS PARA LA PLATAFORMA ROBÓTICA MANFRED.2011
- [5] García. Sistema de Odometría Visual para la Mejora del Posicionamiento Global de un Vehículo.2007
- [6] Sistemas de Locomoción de robots móviles. Recuperado el 25 de agosto 2016 de: http://www.esi2.us.es/~vivas/ayr2iaei/LOC_MOV.pdf
- [7] Quintero. MODELO CINEMÁTICO DINÁMICO DEL MINI ROBÓT MÓVIL RICIMAF. 2012
- [8] Instituto de Investigación Tecnológica. Sensores para robot móviles. Recuperado el 16 de agosto 2016, de: <http://www.iit.comillas.edu/~alvaro/teaching/Clases/Robots/teoria/Sensores%20y%20actuadores.pdf>
- [9] Definición raspberry pi
- [10] Robologs. Tutorial de Arduino y MPU-6050. Recuperado 15 de agosto 2016, de: <http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/>
- [11] Insed. 3-Axis Digital Compass IC HMC5983. Recuperado 4 de agosto 2016, de: http://www.insed.de/HMC5983_Datasheet_FINAL_2011.pdf
- [12] Rutgers University. User's manual dynamixel Ax.12. Recuperado 15 de julio de 2016, de: <http://hackerspace.cs.rutgers.edu/library/Bioloid/doc/AX-12.pdf>

[13] Atmel. 8-bit Microcontroller with 16K Bytes In-System Programmable Flash. Recuperado 15 de julio de 2016, de: <http://www.atmel.com/Images/2466S.pdf>

[14] Pérez N, Salamanca D. DISEÑO, MODELAMIENTO Y SIMULACIÓN 3D DE UN ROBOT MÓVIL PARA EXPLORACIÓN DE TERRENOS.2009

[15] Moreno A. The I2C Bus Specification. 2004

[16] Modelado del Entorno en Robótica Móvil.

ANEXOS

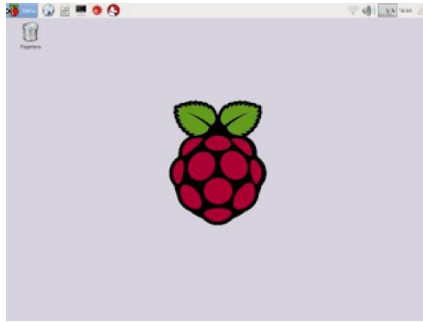
Anexo A Instalación y configuración Raspberry pi:

La instalación del sistema operativo Raspbian Jessie se muestra en los siguientes pasos:

1. Insertar la tarjeta SD, en un lector de tarjetas SD para computador y observar si el computador la lee.
2. Descargar el software Win32DiskImager.
3. Descargar la imagen del sistema operativo Raspbian Jessie de la página oficial de raspberry pi.
4. En el software Win32DiskImager escoger la imagen del sistema operativo Raspbian Jessie y también seleccionar la letra de la unidad en la que quedo registrada la tarjeta SD.
5. Luego hacer clic en escribir y esperar la escritura completa.
6. Salir del software y expulsar la tarjeta SD.

La configuración inicial que se debe hacer a la raspberry pi se presenta en los siguientes pasos:

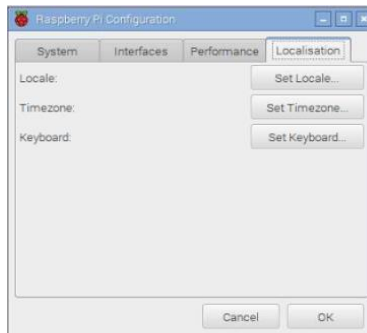
1. Insertar la tarjeta SD en la Raspberry pi.
2. Conectar la Raspberry pi a una pantalla por medio de un cable HDMI, un teclado y un mouse.
3. Alimentar la Raspberry pi con un adaptador a través del conector Micro USB.



*Imagen 47 Visualización escritorio
Raspberry pi*

Fuente: Autor.

Por defecto del sistema operativo el sistema aparecerá en inglés, para comenzar a hacer las configuraciones básicas para esto vamos a Menú-Preferencias-Raspberry pi Configuraciones, aquí se podrá cambiar la hora, localización y cosas del sistema en general.



*Imagen 48 Raspberry pi
Configuraciones*

Fuente: Autor.

Para poder usar la memoria completa de la tarjeta SD ya que la instalación del sistema operativo solo ocupa una parte del tamaño de la tarjeta SD y el otro espacio queda vacío, para poder usar el espacio vacío vamos a configuraciones y en sistema se da clic en expand Filesystem y reiniciamos la Raspberry pi.

Para la comunicación de la Raspberry pi con el computador se hizo mediante un protocolo de comunicación llamado SSH, para esto se utilizó

un adaptador wifi que sea compatible con la Raspberry pi, en este caso mediante el adaptador wifi netis 2120, se muestra a continuación los pasos para configurar la raspberry pi con el computador:

- 1- La primera prueba es conectar las netis a la raspberry y comprobar que se conecte a una red wifi, para poder determinar si este adaptador sirve.
- 2- Se colocan las siguientes instrucciones en un terminal de la raspberry pi.

```
sudo apt-get update
sudo apt-get install hostapd isc-dhcp-server
sudo nano /etc/dhcp/dhcpd.conf
```

Encontrar las siguientes líneas y comentariarlas:

```
option domain-name "example.org";
option          domain-name-servers          ns1.example.org,
ns2.example.org;
```

Encontrar las siguientes líneas y quitar el #:

```
# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
#authoritative;
```

Agregar al final del scroll:

```
subnet 192.168.42.0 netmask 255.255.255.0 {
range 192.168.42.10 192.168.42.50;
option broadcast-address 192.168.42.255;
option routers 192.168.42.1;
default-lease-time 600;
max-lease-time 7200;
option domain-name "local";
option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

Se guardan los cambios y en nuevo terminal ejecutamos la siguiente instrucción:

```
sudo nano /etc/default/isc-dhcp-serv
```

Al final del archivo agregamos la siguiente línea:

```
INTERFACES="wlan0"
```

Se guarda el archivo, ahora para establecer una WLAN para una IP estática se digita la siguiente línea:

```
sudo nano /etc/network/interfaces
```

Se agregan las líneas.

```
iface wlan0 inet static  
address 192.168.42.1  
netmask 255.255.255.0
```

En este caso la IP es 192.168.0.1 , el siguiente paso es ejecutar el `allow-hotplug wlan0` el cual muestra un archivo con la configuración ya realizada.

Ahora se asigna una dirección IP estática al adaptador wifi.

```
sudo ifconfig wlan0 192.168.42.1
```

- 3- Para poder configurar la raspberry como un Access point se ejecutó los siguientes pasos.

Ya se pueden configurar los detalles del punto de acceso, se establece una red protegida por contraseña, se crea un archivo nuevo con el siguiente comando.

```
sudo nano /etc/hostapd/hostapd.conf
```

Se colocan las siguientes líneas de código.

```
interface=wlan0  
driver=rtl871xdrv  
ssid=Pi_AP  
hw_mode=g  
channel=6  
macaddr_acl=0  
auth_algs=1  
ignore_broadcast_ssid=0
```

```
wpa=2  
wpa_passphrase=Raspberry  
wpa_key_mgmt=WPA-PSK  
wpa_pairwise=TKIP  
rsn_pairwise=CCMP
```

El driver que se usó para el adaptador wifi de la netis fue el nl80211, se guarda el archivo y el siguiente paso que se realizo fue decirle a la Raspberry Pi donde encontrar el archivo que se creó con la siguiente línea.

```
sudo nano /etc/default/hostapd  
DAEMON_CONF="/etc/hostapd/hostapd.conf
```

Anexo B Código Sensor Ultrasonido en c

```
#include <Wire.h>
#include <stdint.h>
#include <LiquidCrystal.h>
#include <NewPing.h>
/*
 * Definitions
 */
#define RS 31
#define EN 30
#define BIT4 29
#define BIT5 28
#define BIT6 27
#define BIT7 26

#define SLAVE_ADDRESS 0x42 // I2C slave address

#define SONAR_NUM 5 // Number of sonar sensors

#define ECHO_0 1 // Echo Pin US 0
#define TRIG_0 0 // Trigger Pin US 0

#define ECHO_1 3 // Echo Pin US 1
#define TRIG_1 2 // Trigger Pin US 1

#define ECHO_2 6 // Echo Pin US 1
#define TRIG_2 5 // Trigger Pin US 1

#define ECHO_3 12 // Echo Pin US 0
#define TRIG_3 11 // Trigger Pin US 0

#define ECHO_4 14 // Echo Pin US 0
#define TRIG_4 13 // Trigger Pin US 0

#define MAX_RANGE 100 // Max Range of the sensors
#define MIN_RANGE 0 // Min Range of the sensors

/*
 * Variables
```

```

*/
LiquidCrystal lcd(RS, EN, BIT4, BIT5, BIT6, BIT7);

uint8_t state = 0;
uint8_t distance;

NewPing sonar[SONAR_NUM] = { // Sensor object array.
  NewPing(TRIG_0, ECHO_0, MAX_RANGE), // Each sensor's trigger
  pin, echo pin, and max distance to ping.
  NewPing(TRIG_1, ECHO_1, MAX_RANGE),
  NewPing(TRIG_2, ECHO_2, MAX_RANGE),
  NewPing(TRIG_3, ECHO_3, MAX_RANGE),
  NewPing(TRIG_4, ECHO_4, MAX_RANGE)
};

uint8_t distSensorBufferA[5] = {0, 0, 0, 0, 0};
uint8_t distSensorBufferB[5] = {0, 0, 0, 0, 0};

/*
 * Functions or Methods
 */

uint8_t readDataSensor(uint8_t sensorV0);
void printData(uint8_t *data);
uint8_t getDistanceUS(uint8_t trg, uint8_t echo);
/*
 * Program
 */

void setup(){
  lcd.begin(16, 2);
  lcd.clear();
  lcd.print("Distance in cm:");

  pinMode(TRIG_0, OUTPUT);
  pinMode(ECHO_0, INPUT);
  pinMode(TRIG_1, OUTPUT);
  pinMode(ECHO_1, INPUT);
  pinMode(TRIG_2, OUTPUT);
  pinMode(ECHO_2, INPUT);
  pinMode(TRIG_3, OUTPUT);

```

```

pinMode(ECHO_3, INPUT);
pinMode(TRIG_4, OUTPUT);
pinMode(ECHO_4, INPUT);

Wire.begin(SLAVE_ADDRESS); // join i2c bus with address #0x42
Wire.onReceive(receiveEvent); // register event
Wire.onRequest(sendEvent);
Serial.begin(9600);
}

void loop(){

    delay(200);
    distSensorBufferA[0] = sonar[0].ping_cm();
    distSensorBufferA[1] = sonar[1].ping_cm();
    distSensorBufferA[2] = sonar[2].ping_cm();
    distSensorBufferA[3] = sonar[3].ping_cm();
    distSensorBufferA[4] = sonar[4].ping_cm();

    printData(&distSensorBufferA[0]);
}

void receiveEvent(int howMany) {
    //lcd.setCursor(0, 0);
    //lcd.print("Ilego: ");
    //while (1 <= Wire.available()) { // loop through all but the last
        state = Wire.read(); // receive byte as a character
    //lcd.print(state);
    }

void sendEvent(){
    //Serial.println(distance);
    if (state == 0x01){
        Wire.write((uint8_t)distSensorBufferA[0]);
    }
    else if (state == 0x02){
        Wire.write((uint8_t)distSensorBufferA[1]);
    }
    else if (state == 0x03){
        Wire.write((uint8_t)distSensorBufferA[2]);
    }
}

```

```

    }
    else if (state == 0x04){
        Wire.write((uint8_t)distSensorBufferA[3]);
    }
    else if (state == 0x05){
        Wire.write((uint8_t)distSensorBufferA[4]);
    }

    state = 0;
}

uint8_t readDataSensor(uint8_t sensorV0){
    float volts = analogRead(sensorV0)*0.0048828125;    // value from
    sensor * (5/1024) - if running 3.3.volts then change 5 to 3.3
    float distance = 28.4*pow(volts, -1.42);
    if (distance <= 80) {
        return (uint8_t)distance;
    }
    else return 80;
}

void printData(uint8_t *data){
    lcd.setCursor(0, 1);
    lcd.print("          ");
    lcd.setCursor(0, 1);

    uint8_t i = 0;
    for (i = 0; i<4; i++){
        lcd.print(*data);
        lcd.setCursor(2+i*3, 1);
        lcd.print(",");
        data++;
    }
    lcd.print(*data);
}

```


Anexo C Librería sensor MPU6050

```
import smbus

class mpu6050:

    # Global Variables
    GRAVITY_MS2 = 9.80665
    address = None
    bus = smbus.SMBus(1)

    # Scale Modifiers
    ACCEL_SCALE_MODIFIER_2G = 16384.0
    ACCEL_SCALE_MODIFIER_4G = 8192.0
    ACCEL_SCALE_MODIFIER_8G = 4096.0
    ACCEL_SCALE_MODIFIER_16G = 2048.0

    GYRO_SCALE_MODIFIER_250DEG = 131.0
    GYRO_SCALE_MODIFIER_500DEG = 65.5
    GYRO_SCALE_MODIFIER_1000DEG = 32.8
    GYRO_SCALE_MODIFIER_2000DEG = 16.4

    # Pre-defined ranges
    ACCEL_RANGE_2G = 0x00
    ACCEL_RANGE_4G = 0x08
    ACCEL_RANGE_8G = 0x10
    ACCEL_RANGE_16G = 0x18

    GYRO_RANGE_250DEG = 0x00
    GYRO_RANGE_500DEG = 0x08
    GYRO_RANGE_1000DEG = 0x10
    GYRO_RANGE_2000DEG = 0x18

    # MPU-6050 Registers
    PWR_MGMT_1 = 0x6B
    PWR_MGMT_2 = 0x6C

    SELF_TEST_X = 0x0D
    SELF_TEST_Y = 0x0E
    SELF_TEST_Z = 0x0F
```

SELF_TEST_A = 0x10

ACCEL_XOUT0 = 0x3B

ACCEL_XOUT1 = 0x3C

ACCEL_YOUT0 = 0x3D

ACCEL_YOUT1 = 0x3E

ACCEL_ZOUT0 = 0x3F

ACCEL_ZOUT1 = 0x40

TEMP_OUT0 = 0x41

TEMP_OUT1 = 0x42

GYRO_XOUT0 = 0x43

GYRO_XOUT1 = 0x44

GYRO_YOUT0 = 0x45

GYRO_YOUT1 = 0x46

GYRO_ZOUT0 = 0x47

GYRO_ZOUT1 = 0x48

ACCEL_CONFIG = 0x1C

GYRO_CONFIG = 0x1B

```
def __init__(self, address):
```

```
    self.address = address
```

```
    # Wake up the MPU-6050 since it starts in sleep mode
```

```
    self.bus.write_byte_data(self.address, self.PWR_MGMT_1, 0x00)
```

```
# I2C communication methods
```

```
def read_i2c_word(self, register):
```

```
    """Read two i2c registers and combine them.
```

```
    Register – the first register to read from.
```

```
    Returns the combined read results.
```

```
    """
```

```
    # Read the data from the registers
```

```
    high = self.bus.read_byte_data(self.address, register)
```

```
    low = self.bus.read_byte_data(self.address, register + 1)
```

```
    value = (high << 8) + low
```

```

    if (value >= 0x8000):
        return -((65535 - value) + 1)
    else:
        return value

# MPU-6050 Methods

def get_temp(self):
    """Reads the temperature from the onboard temperature
    sensor of the MPU-6050.
    Returns the temperature in degrees Celcius.
    """
    raw_temp = self.read_i2c_word(self.TEMP_OUT0)

    # Get the actual temperature using the formule given in the
    # MPU-6050 Register Map and Descriptions section 4.2, page
30    actual_temp = (raw_temp / 340.0) + 36.53

    return actual_temp

def set_accel_range(self, accel_range):
    """Sets the range of the accelerometer to range.
    Accel_range – the range to set the accelerometer to. Using a
    pre-defined range is advised.
    """
    # First change it to 0x00 to make sure we write the correct value later
    self.bus.write_byte_data(self.address, self.ACCEL_CONFIG, 0x00)

    # Write the new range to the ACCEL_CONFIG register
    self.bus.write_byte_data(self.address, self.ACCEL_CONFIG,
    accel_range)

def read_accel_range(self, raw = False):
    """Reads the range the accelerometer is set to.
    If raw is True, it will return the raw value from the ACCEL_CONFIG
    register
    If raw is False, it will return an integer: -1, 2, 4, 8 or 16. When it
    returns -1 something went wrong.
    """

```

```

raw_data = self.bus.read_byte_data(self.address,
self.ACCEL_CONFIG)

```

```

if raw is True:
    return raw_data
elif raw is False:
    if raw_data == self.ACCEL_RANGE_2G:
        return 2
    elif raw_data == self.ACCEL_RANGE_4G:
        return 4
    elif raw_data == self.ACCEL_RANGE_8G:
        return 8
    elif raw_data == self.ACCEL_RANGE_16G:
        return 16
    else:
        return -1

```

```

def get_accel_data(self, g = False):
    """Gets and returns the X, Y and Z values from the accelerometer.
    If g is True, it will return the data in g
    If g is False, it will return the data in m/s^2
    Returns a dictionary with the measurement results.
    """

```

```

x = self.read_i2c_word(self.ACCEL_XOUT0)
y = self.read_i2c_word(self.ACCEL_YOUT0)
z = self.read_i2c_word(self.ACCEL_ZOUT0)

```

```

accel_scale_modifier = None
accel_range = self.read_accel_range(True)

```

```

if accel_range == self.ACCEL_RANGE_2G:
    accel_scale_modifier = self.ACCEL_SCALE_MODIFIER_2G
elif accel_range == self.ACCEL_RANGE_4G:
    accel_scale_modifier = self.ACCEL_SCALE_MODIFIER_4G
elif accel_range == self.ACCEL_RANGE_8G:
    accel_scale_modifier = self.ACCEL_SCALE_MODIFIER_8G
elif accel_range == self.ACCEL_RANGE_16G:
    accel_scale_modifier = self.ACCEL_SCALE_MODIFIER_16G
else:
    print("Unkown range - accel_scale_modifier set to
self.ACCEL_SCALE_MODIFIER_2G")

```

```

        accel_scale_modifier = self.ACCEL_SCALE_MODIFIER_2G

    x = x / accel_scale_modifier
    y = y / accel_scale_modifier
    z = z / accel_scale_modifier

    if g is True:
        return {'x': x, 'y': y, 'z': z}
    elif g is False:
        x = x * self.GRAVITY_MS2
        y = y * self.GRAVITY_MS2
        z = z * self.GRAVITY_MS2
        return {'x': x, 'y': y, 'z': z}

def set_gyro_range(self, gyro_range):
    """Sets the range of the gyroscope to range.
    Gyro_range – the range to set the gyroscope to. Using a pre-defined
    range is advised.
    """
    # First change it to 0x00 to make sure we write the correct value later
    self.bus.write_byte_data(self.address, self.GYRO_CONFIG, 0x00)

    # Write the new range to the ACCEL_CONFIG register
    self.bus.write_byte_data(self.address, self.GYRO_CONFIG,
gyro_range)

def read_gyro_range(self, raw = False):
    """Reads the range the gyroscope is set to.
    If raw is True, it will return the raw value from the GYRO_CONFIG
    register.
    If raw is False, it will return 250, 500, 1000, 2000 or -1. If the
    returned value is equal to -1 something went wrong.
    """
    raw_data = self.bus.read_byte_data(self.address,
self.GYRO_CONFIG)

    if raw is True:
        return raw_data
    elif raw is False:
        if raw_data == self.GYRO_RANGE_250DEG:
            return 250

```

```

        elif raw_data == self.GYRO_RANGE_500DEG:
            return 500
        elif raw_data == self.GYRO_RANGE_1000DEG:
            return 1000
        elif raw_data == self.GYRO_RANGE_2000DEG:
            return 2000
        else:
            return -1

def get_gyro_data(self):
    """Gets and returns the X, Y and Z values from the gyroscope.
    Returns the read values in a dictionary.
    """
    x = self.read_i2c_word(self.GYRO_XOUT0)
    y = self.read_i2c_word(self.GYRO_YOUT0)
    z = self.read_i2c_word(self.GYRO_ZOUT0)

    gyro_scale_modifier = None
    gyro_range = self.read_gyro_range(True)

    if gyro_range == self.GYRO_RANGE_250DEG:
        gyro_scale_modifier = self.GYRO_SCALE_MODIFIER_250DEG
    elif gyro_range == self.GYRO_RANGE_500DEG:
        gyro_scale_modifier = self.GYRO_SCALE_MODIFIER_500DEG
    elif gyro_range == self.GYRO_RANGE_1000DEG:
        gyro_scale_modifier = self.GYRO_SCALE_MODIFIER_1000DEG
    elif gyro_range == self.GYRO_RANGE_2000DEG:
        gyro_scale_modifier = self.GYRO_SCALE_MODIFIER_2000DEG
    else:
        print("Unkown range - gyro_scale_modifier set to self.GYRO_SCALE_MODIFIER_250DEG")
        gyro_scale_modifier = self.GYRO_SCALE_MODIFIER_250DEG

    x = x / gyro_scale_modifier
    y = y / gyro_scale_modifier
    z = z / gyro_scale_modifier

    return {'x': x, 'y': y, 'z': z}

```

```

def get_all_data(self):
    """Reads and returns all the available data."""
    Temp = get_temp()
    accel = get_accel_data()
    gyro = get_gyro_data()

    return [accel, gyro, temp]

if __name__ == "__main__":
    mpu = MPU6050(0x68)
    print(mpu.get_temp())
    accel_data = mpu.get_accel_data()
    print(accel_data['x'])
    print(accel_data['y'])
    print(accel_data['z'])
    gyro_data = mpu.get_gyro_data()
    print(gyro_data['x'])
    print(gyro_data['y'])
    print(gyro_data['z'])

```

Anexo D liberaría Sensor HCM5983

```
import smbus
import time
import math
import re

#
=====
=====
# HMC5983_I2C Class
#
=====
=====

class HMC5983(object):
    @staticmethod
    def get_bus_I2C():
        # Based from www.adafruit.com
        # Revision list available at:
        http://elinux.org/RPi_HardwareHistory#Board_Revision_History
        try:
            with open('/proc/cpuinfo', 'r') as infile:
                for line in infile:
                    # Match a line of the form "Revision : 0002" while ignoring
                    extra
                    # info in front of the revision (like 1000 when the Pi was over-
                    volted).
                    match = re.match('Revision\s+:\s+.*(\w{4})$', line)
                    if match and match.group(1) in ['0000', '0002', '0003']:
                        # Return revision 1 if revision ends with 0000, 0002 or
                        0003.
                        return 1
                    elif match:
                        # Assume revision 2 if revision ends with any other 4
                        chars.
                        return 2
                    # Couldn't find the revision, assume revision 0 like older code
                    for compatibility.
```



```

        return 0
    except:
        return 0

__CON_REG_A = 0x00
__CON_REG_B = 0x01
__MODE_REGISTER = 0x02
__AXIS_X_MSB = 0x03
__AXIS_X_LSB = 0x04
__AXIS_Z_MSB = 0x05
__AXIS_Z_LSB = 0x06
__AXIS_Y_MSB = 0x07
__AXIS_Y_LSB = 0x08
__STATUS_REGISTER = 0x09
__IDENTIFICATION_REG_A = 0x0A
__IDENTIFICATION_REG_B = 0x0B
__IDENTIFICATION_REG_C = 0x0C
__TEMPERATURE_MSB = 0x31
__TEMPERATURE_LSB = 0x32

__CONTINUOUS_MODE = 0x00
__SINGLE_MODE = 0x01
__IDLE_MODE = 0x02

def __init__(self, bus_I2C = 1):
    self.address = 0x1e
    if bus_I2C != 1:
        self.bus = smbus.SMBus(HMC5983.get_bus_I2C())
    else:
        self.bus = smbus.SMBus(1)
    self.scale = 0.92

    # Set 8 samples, temperature on, normal measure and 15hz sample
    self.write_byte(self.__CON_REG_A, 0xf0) # 0xff, 0xa0, 0x10
    # 1.3 gain LSB / Gauss 1090 (default)
    self.write_byte(self.__CON_REG_B, 0x20) # 0xff, 0x20
    # Set continuous mode
    self.write_byte(self.__MODE_REGISTER,
self.__CONTINUOUS_MODE)

def read_byte(self, adr):

```

```

        return self.bus.read_byte_data(self.address, adr)

    def read_word(self, adr):
        low = self.bus.read_byte_data(self.address, adr)
        high = self.bus.read_byte_data(self.address, adr - 1)
        val = (high << 8) + low
        return val

    def read_word_2c(self, adr):
        val = self.read_word(adr)
        if val >= 0x8000:
            return -((65535 - val) + 1)
        else:
            return val

    def write_byte(self, adr, value):
        self.bus.write_byte_data(self.address, adr, value)

    def get_axes(self):
        flag = 1
        counter_t_out = 0
        while flag == 1 or counter_t_out > 40:
            axes = []
            try:

                axes.append(self.read_word_2c(self.__AXIS_X_LSB) * self.scale)
            # x_axis

                axes.append(self.read_word_2c(self.__AXIS_Y_LSB) * self.scale)
            # y_axis

                axes.append(self.read_word_2c(self.__AXIS_Z_LSB) * self.scale)
            # z_axis

            #print "axes =", axes
            flag = 0
        except:
            flag = 1
            counter_t_out = counter_t_out + 1
        return axes

    def get_angle(self):

```

```
axes = self.get_axes()
angle = math.atan2(axes[1], axes[0])
if (angle < 0):
    angle += 2 * math.pi
if (angle > 2*math.pi):
    angle -= 2*math.pi
return angle, math.degrees(angle)
```

Anexo E Programación de ATmega16 en Arduino IDE.

Para la programación del atmega16 sobre la plataforma de arduino IDE, se debe realizar los siguientes pasos, Todo se realizó sobre Windows.

- 1- En el computador se dirige a la siguiente dirección : Windows (C:) / Program Files(x86) / Arduino / hardware / arduino / avr / variants

Se va a crear una carpeta, se tiene que tener en cuenta el nombre de la carpeta ya que el programa va a ir a buscar en el nombre específico de la carpeta en este caso se creó la carpeta con el nombre “atmega166”, en esta carpeta se pegara el siguiente archivo con nombre pins_arduino.h

```
#ifndef Pins_Arduino_h65
#define Pins_Arduino_h

#include <avr/pgmspace.h>

#define NUM_DIGITAL_PINS      32
#define NUM_ANALOG_INPUTS     8
#define      analogInputToDigitalPin(p)      ((p      <
NUM_ANALOG_INPUTS) ? (p) + 24 : -1)
//#define digitalPinHasPWM(p)      ((p) == 3 || (p) == 12 || (p) ==
13 || (p) == 15)
#define digitalPinHasPWM(p)      ((p) == 12 || (p) == 13 || (p) ==
15)

static const uint8_t SS  = 4;
static const uint8_t MOSI = 5;
static const uint8_t MISO = 6;
static const uint8_t SCK = 7;

static const uint8_t SDA = 17;
static const uint8_t SCL = 16;

#define LED_BUILTIN 7
```

```

static const uint8_t A0 = 31;
static const uint8_t A1 = 30;
static const uint8_t A2 = 29;
static const uint8_t A3 = 28;
static const uint8_t A4 = 27;
static const uint8_t A5 = 26;
static const uint8_t A6 = 25;
static const uint8_t A7 = 24;

#define digitalPinToPCICR(p) (((p) >= 0 && (p) <
NUM_DIGITAL_PINS) ? (&PCICR) : ((uint8_t *)0))
#define digitalPinToPCICRbit(p) (((p) <= 7) ? 1 : (((p) <= 15) ? 3 :
(((p) <= 23) ? 2 : 0)))
#define digitalPinToPCMSK(p) (((p) <= 7) ? (&PCMSK2) : (((p)
<= 13) ? (&PCMSK0) : (((p) <= 21) ? (&PCMSK1) : ((uint8_t *)0))))
#define digitalPinToPCMSKbit(p) ((p) % 8)
#define digitalPinToInterrupt(p) ((p) == 10 ? 0 : ((p) == 11 ? 1 :
((p) == 2 ? 2 : NOT_AN_INTERRUPT)))

#ifndef ARDUINO_MAIN

const uint16_t PROGMEM port_to_mode_PGM[] =
{
    NOT_A_PORT,
    (uint16_t) &DDRA,
    (uint16_t) &DDRB,
    (uint16_t) &DDRC,
    (uint16_t) &DDRD,
};

const uint16_t PROGMEM port_to_output_PGM[] =
{
    NOT_A_PORT,
    (uint16_t) &PORTA,
    (uint16_t) &PORTB,
    (uint16_t) &PORTC,
    (uint16_t) &PORTD,
};

const uint16_t PROGMEM port_to_input_PGM[] =

```

```

{
    NOT_A_PORT,
    (uint16_t) &PINA,
    (uint16_t) &PINB,
    (uint16_t) &PINC,
    (uint16_t) &PIND,
};

```

```

const uint8_t PROGMEM digital_pin_to_port_PGM[32] = {
    PB, // PB0 ** D0
    PB, // PB1 ** D1
    PB, // PB2 ** D2
    PB, // PB3 ** D3
    PB, // PB4 ** D4
    PB, // PB5 ** D5
    PB, // PB6 ** D6
    PB, // PB7 ** D7
    PD, // PD0 ** D8
    PD, // PD1 ** D9
    PD, // PD2 ** D10
    PD, // PD3 ** D11
    PD, // PD4 ** D12
    PD, // PD5 ** D13
    PD, // PD6 ** D14
    PD, // PD7 ** D15
    PC, // PC0 ** D16
    PC, // PC1 ** D17
    PC, // PC2 ** D18
    PC, // PC3 ** D19
    PC, // PC4 ** D20
    PC, // PC5 ** D21
    PC, // PC6 ** D22
    PC, // PC7 ** D23
    PA, // PA7 ** A7 D24
    PA, // PA6 ** A6 D25
    PA, // PA5 ** A5 D26
    PA, // PA4 ** A4 D27
    PA, // PA3 ** A3 D28
    PA, // PA2 ** A2 D29
    PA, // PA1 ** A1 D30
    PA, // PA0 ** A0 D31

```

```

};

const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[32] = {

    _BV(0), // PB0 ** D0
    _BV(1), // PB1 ** D1
    _BV(2), // PB2 ** D2
    _BV(3), // PB3 ** D3
    _BV(4), // PB4 ** D4
    _BV(5), // PB5 ** D5
    _BV(6), // PB6 ** D6
    _BV(7), // PB7 ** D7
    _BV(0), // PD0 ** D8
    _BV(1), // PD1 ** D9
    _BV(2), // PD2 ** D10
    _BV(3), // PD3 ** D11
    _BV(4), // PD4 ** D12
    _BV(5), // PD5 ** D13
    _BV(6), // PD6 ** D14
    _BV(7), // PD7 ** D15
    _BV(0), // PC0 ** D16
    _BV(1), // PC1 ** D17
    _BV(2), // PC2 ** D18
    _BV(3), // PC3 ** D19
    _BV(4), // PC4 ** D20
    _BV(5), // PC5 ** D21
    _BV(6), // PC6 ** D22
    _BV(7), // PC7 ** D23
    _BV(7), // PA7 ** A7 D24
    _BV(6), // PA6 ** A6 D25
    _BV(5), // PA5 ** A5 D26
    _BV(4), // PA4 ** A4 D27
    _BV(3), // PA3 ** A3 D28
    _BV(2), // PA2 ** A2 D29
    _BV(1), // PA1 ** A1 D30
    _BV(0), // PA0 ** A0 D31
};

const uint8_t PROGMEM digital_pin_to_timer_PGM[] =
{
    NOT_ON_TIMER, /* 0 - PB0 */

```

```

NOT_ON_TIMER, /* 1 - PB1 */
NOT_ON_TIMER, /* 2 - PB2 */
TIMER0A,      /* 3 - PB3 */
// TIMER0,     /* 3 - PB3 */
NOT_ON_TIMER, /* 4 - PB4 */
NOT_ON_TIMER, /* 5 - PB5 */
NOT_ON_TIMER, /* 6 - PB6 */
NOT_ON_TIMER, /* 7 - PB7 */
NOT_ON_TIMER, /* 8 - PD0 */
NOT_ON_TIMER, /* 9 - PD1 */
NOT_ON_TIMER, /* 10 - PD2 */
NOT_ON_TIMER, /* 11 - PD3 */
TIMER1B,      /* 12 - PD4 */
TIMER1A,      /* 13 - PD5 */
NOT_ON_TIMER, /* 14 - PD6 */
TIMER2,       /* 15 - PD7 */
NOT_ON_TIMER, /* 16 - PC0 */
NOT_ON_TIMER, /* 17 - PC1 */
NOT_ON_TIMER, /* 18 - PC2 */
NOT_ON_TIMER, /* 19 - PC3 */
NOT_ON_TIMER, /* 20 - PC4 */
NOT_ON_TIMER, /* 21 - PC5 */
NOT_ON_TIMER, /* 22 - PC6 */
NOT_ON_TIMER, /* 23 - PC7 */
NOT_ON_TIMER, /* 24 - PA7 */
NOT_ON_TIMER, /* 25 - PA6 */
NOT_ON_TIMER, /* 26 - PA5 */
NOT_ON_TIMER, /* 27 - PA4 */
NOT_ON_TIMER, /* 28 - PA3 */
NOT_ON_TIMER, /* 29 - PA2 */
NOT_ON_TIMER, /* 30 - PA1 */
NOT_ON_TIMER /* 31 - PA0 */
};

```

```

#endif

```

```

// These serial port names are intended to allow libraries and
architecture-neutral
// sketches to automatically default to the correct port name for a
particular type
// of use. For example, a GPS module would normally connect to
SERIAL_PORT_HARDWARE_OPEN,

```



```

// the first hardware serial port whose RX/TX pins are not dedicated
// to another use.
//
// SERIAL_PORT_MONITOR      Port which normally prints to the
// Arduino Serial Monitor
//
// SERIAL_PORT_USBVIRTUAL   Port which is USB virtual serial
//
// SERIAL_PORT_LINUXBRIDGE  Port which connects to a Linux
// system via Bridge library
//
// SERIAL_PORT_HARDWARE     Hardware serial port, physical
// RX & TX pins.
//
// SERIAL_PORT_HARDWARE_OPEN Hardware serial ports
// which are open for use. Their RX & TX
// pins are NOT connected to anything by default.
#define SERIAL_PORT_MONITOR Serial
#define SERIAL_PORT_HARDWARE Serial

#endif

```

```

ATMEGA16

+---\ /---+
(D 0) PB0 1|      |40 PA0 (AI 0 / D31)
(D 1) PB1 2|      |39 PA1 (AI 1 / D30)
INT2 (D 2) PB2 3|    |38 PA2 (AI 2 / D29)
PWM (D 3) PB3 4|    |37 PA3 (AI 3 / D28)
PWM/SS (D 4) PB4 5|  |36 PA4 (AI 4 / D27)
MOSI (D 5) PB5 6|    |35 PA5 (AI 5 / D26)
PWM/MISO (D 6) PB6 7| |34 PA6 (AI 6 / D25)
PWM/SCK (D 7) PB7 8| |33 PA7 (AI 7 / D24)
RST 9|      |32 AREF
VCC 10|     |31 GND
GND 11|     |30 AVCC
XTAL2 12|    |29 PC7 (D 23)
XTAL1 13|    |28 PC6 (D 22)
RX0 (D 8) PD0 14|   |27 PC5 (D 21) TDI
TX0 (D 9) PD1 15|   |26 PC4 (D 20) TDO
RX1/INT0 (D 10) PD2 16| |25 PC3 (D 19) TMS
TX1/INT1 (D 11) PD3 17| |24 PC2 (D 18) TCK
PWM (D 12) PD4 18|   |23 PC1 (D 17) SDA
PWM (D 13) PD5 19|   |22 PC0 (D 16) SCL
(D 14) PD6 20|   |21 PD7 (D 15) PWM
+-----+

```

Este archivo le indica al arduino los números de pines digitales, analógicos y configura nuestro atmega 16 como se muestra en la imagen anterior.

- 2- El siguiente paso es ir a la siguiente dirección : Windows (C:) / Program Files(x86) / Arduino / hardware / arduino / avr /

Buscar el archivo boards.txt y al final de este copiar lo siguiente:

```
#####  
#####  
atmega16-8.name=Atmega16 (internal 8MHz clock)  
  
atmega16-8.upload.tool=avrdude  
atmega16-8.upload.protocol=stk500v1  
atmega16-8.upload.maximum_size=14336  
atmega16-8.upload.speed=19200  
  
atmega16-8.bootloader.low_fuses=0x84  
atmega16-8.bootloader.high_fuses=0x99  
atmega16-8.bootloader.tool=avrdude  
  
atmega16-8.build.mcu=atmega16  
atmega16-8.build.f_cpu=8000000L  
atmega16-8.build.core=arduino:arduino  
atmega16-8.build.variant=Atmega166  
  
#####  
#####  
# TODO: Add definitions for ATmega external clock
```

Si nos damos cuenta en la parte donde dice atmega16-8.build.variant= se coloca el nombre de la carpeta que se creó anteriormente.

- 3- El siguiente paso es ir a la siguiente dirección: Windows (C:) / Program Files(x86) / Arduino / hardware / arduino / avr / cores / arduino

Abrir el archivo **HardwareSerial.cpp**, buscar las siguientes líneas:

```
#if defined(__AVR_ATmega8__)  
    config |= 0x80; // select UCSRC register (shared with UBRRH)  
#endif
```

Y cambiarlas por estas:

```
#if defined(__AVR_ATmega8__) || defined(__AVR_ATmega32__) ||  
defined(__AVR_ATmega16__)  
    config |= 0x80; // select UCSRC register (shared with UBRRH)  
#endif
```

4- Para rectificar los pasos anteriores se realizó lo siguiente:

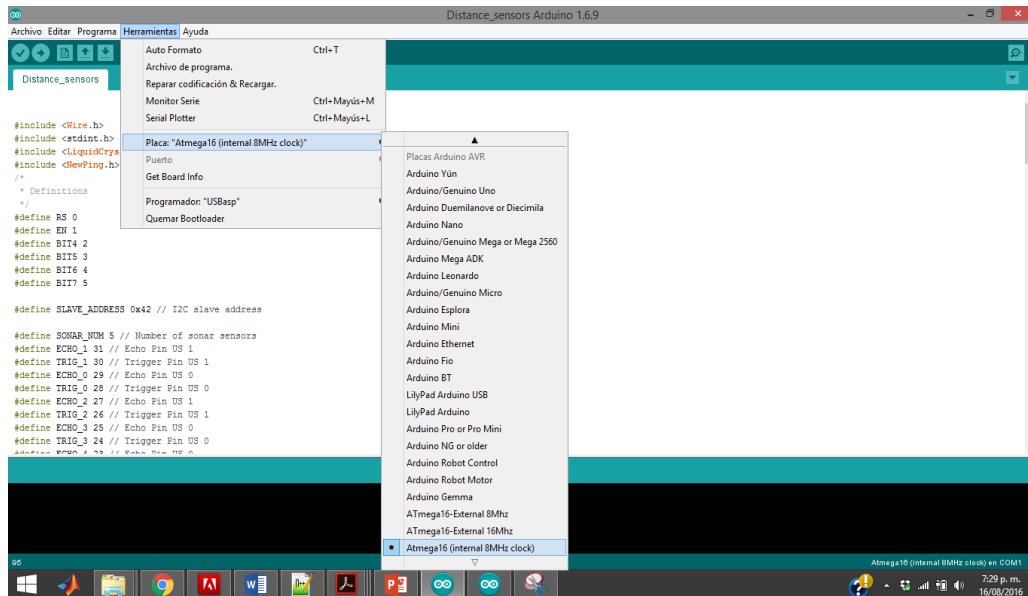


Imagen 49 Selección en el programa del uC Atmega16

Fuente: Autor.

Y para poder programar nuestro Atmega16 se utilizó USBASP v2.0 AVR Programmer, se seleccionó en el arduino IDE.

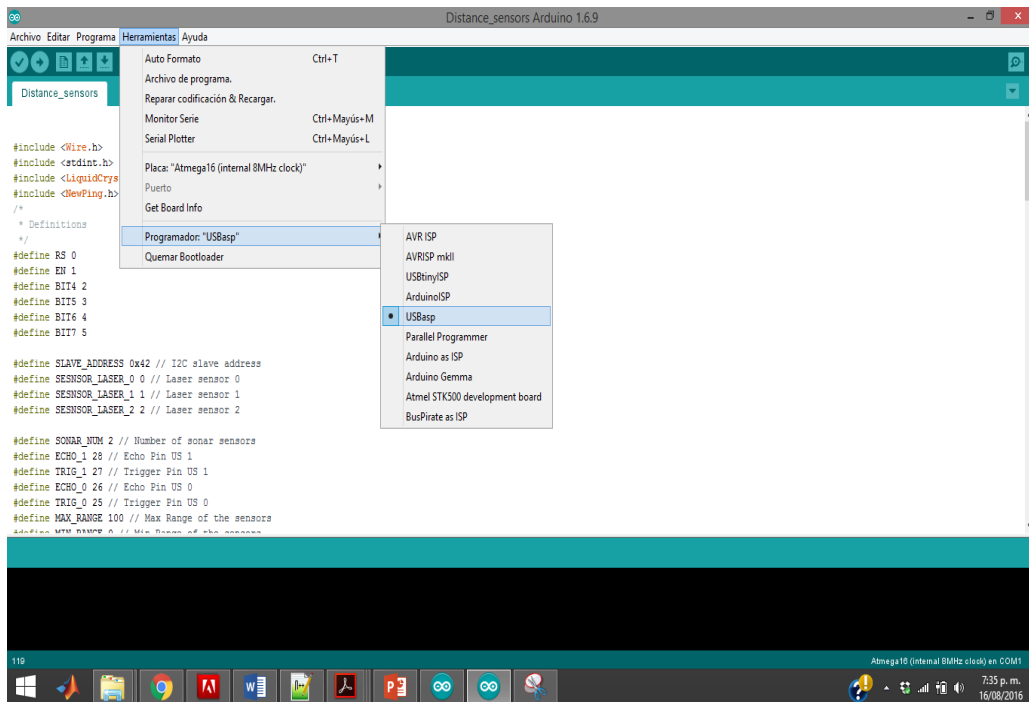


Imagen 50 Selección de Programador

Fuente: Autor.

Ya al haber escogido el programador, para poder subir el programa al atmega16 se realizó de la siguiente manera.

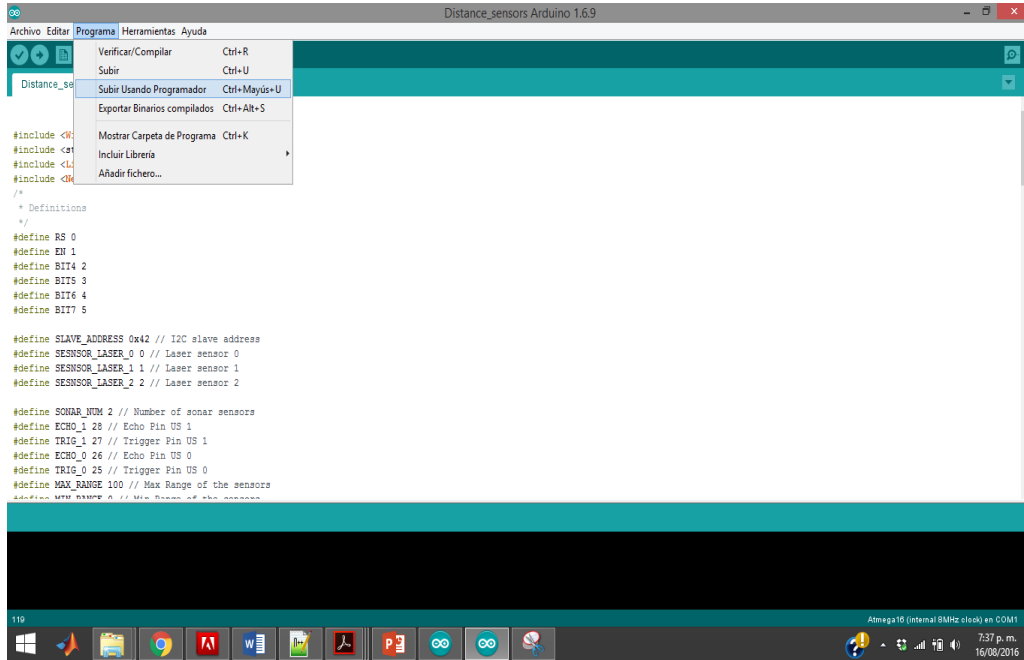


Imagen 51 Cargar el Programa a el ATmega16

Fuente: Autor.

Anexo F Script instalación OPENCV en Raspberry pi

```
cd ~
sudo apt-get update
sudo apt-get upgrade
version="$(wget -q -O - http://sourceforge.net/projects/opencvlibrary/files/opencv-unix | egrep -
m1 -o \"[0-9](\\.[0-9]+)+\" | cut -c2-)"
echo "Installing OpenCV" $version
mkdir OpenCV
cd OpenCV

echo "Installing Dependences"

sudo apt-get install git

sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev libpng12-dev

sudo apt-get install libgtk2.0-dev

sudo apt-get install libatlas-base-dev gfortran

wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py
pip install numpy
sudo apt-get install python2.7-dev

sudo apt-get -qq install libopencv-dev build-essential checkinstall cmake
pkg-config yasm libjpeg-dev libavcodec-dev libavformat-dev libswscale-
dev libdc1394-22-dev libxine-dev libgstreamer0.10-dev libgstreamer-
plugins-base0.10-dev libv4l-dev python-dev python-numpy libtbb-dev
libqt4-dev libfaac-dev libmp3lame-dev libopencore-amrnb-dev
libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev x264
v4l-utils qt5-default checkinstall

echo "Downloading OpenCV" $version

cd ~
```

```

git clone https://github.com/Itseez/opencv_contrib.git
cd opencv_contrib
git checkout version
cd ~
wget -O OpenCV-$version.zip
http://sourceforge.net/projects/opencvlibrary/files/opencv-
unix/$version/opencv-"$version".zip/download

echo "Installing OpenCV" $version
unzip OpenCV-$version.zip
cd opencv-$version
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE -D
CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D
BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D
INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON
-D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON ..
make -j4
sudo make install
sudo ldconfig
sudo sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/opencv.conf'
sudo ldconfig
echo "OpenCV" $version "ready to be used"

```


Anexo F Primer ejemplo OPENCV

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):

    # Take each frame
    _, frame = cap.read()

    # Convert BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # define range of blue color in HSV
    lower_blue = np.array([110,50,50])
    upper_blue = np.array([130,255,255])

    # Threshold the HSV image to get only blue colors
    mask = cv2.inRange(hsv, lower_blue, upper_blue)

    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(frame,frame, mask= mask)

    cv2.imshow('frame',frame)
    cv2.imshow('mask',mask)
    cv2.imshow('res',res)
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
```

Anexo G Código Calibración de la Cámara

```
import numpy as np
import cv2
import glob

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....., (6,5,0)
objp = np.zeros((6*7,3), np.float32)
objp[:, :2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob('img_*.jpg')

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (7,6),None)

    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (7,6), corners2,ret)
        cv2.imshow('img',img)

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1],None,None)
#dist = np.array([-0.13615181, 0.53005398, 0, 0, 0])
```

```

img = cv2.imread('img_1.jpg')
h, w = img.shape[:2]
print "MTX",mtx,type(mtx)
print "DSIT",dist,type(dist)
newcameramt, roi =
cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),1,(w,h))
dst = cv2.undistort(img, mtx, dist, None, newcameramt)
print roi
print newcameramt
# crop the image
x,y,w,h = roi
dst = dst[y:y+h, x:x+w]
cv2.imshow("ds",dst)
cv2.imwrite('calibresultttt.png',dst)

while(1):
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()

```

Anexo H Librería motores Dynamixel

```
from time import sleep
from serial import Serial
import RPi.GPIO as GPIO

class Ax12:
    # important AX-12 constants
    # //////////////////////////////////// EEPROM AREA
    AX_MODEL_NUMBER_L = 0
    AX_MODEL_NUMBER_H = 1
    AX_VERSION = 2
    AX_ID = 3
    AX_BAUD_RATE = 4
    AX_RETURN_DELAY_TIME = 5
    AX_CW_ANGLE_LIMIT_L = 6
    AX_CW_ANGLE_LIMIT_H = 7
    AX_CCW_ANGLE_LIMIT_L = 8
    AX_CCW_ANGLE_LIMIT_H = 9
    AX_SYSTEM_DATA2 = 10
    AX_LIMIT_TEMPERATURE = 11
    AX_DOWN_LIMIT_VOLTAGE = 12
    AX_UP_LIMIT_VOLTAGE = 13
    AX_MAX_TORQUE_L = 14
    AX_MAX_TORQUE_H = 15
    AX_RETURN_LEVEL = 16
    AX_ALARM_LED = 17
    AX_ALARM_SHUTDOWN = 18
    AX_OPERATING_MODE = 19
    AX_DOWN_CALIBRATION_L = 20
    AX_DOWN_CALIBRATION_H = 21
    AX_UP_CALIBRATION_L = 22
    AX_UP_CALIBRATION_H = 23

    # //////////////////////////////////// RAM AREA
    AX_TORQUE_STATUS = 24
    AX_LED_STATUS = 25
    AX_CW_COMPLIANCE_MARGIN = 26
    AX_CCW_COMPLIANCE_MARGIN = 27
    AX_CW_COMPLIANCE_SLOPE = 28
```

AX_CCW_COMPLIANCE_SLOPE = 29
AX_GOAL_POSITION_L = 30
AX_GOAL_POSITION_H = 31
AX_GOAL_SPEED_L = 32
AX_GOAL_SPEED_H = 33
AX_TORQUE_LIMIT_L = 34
AX_TORQUE_LIMIT_H = 35
AX_PRESENT_POSITION_L = 36
AX_PRESENT_POSITION_H = 37
AX_PRESENT_SPEED_L = 38
AX_PRESENT_SPEED_H = 39
AX_PRESENT_LOAD_L = 40
AX_PRESENT_LOAD_H = 41
AX_PRESENT_VOLTAGE = 42
AX_PRESENT_TEMPERATURE = 43
AX_REGISTERED_INSTRUCTION = 44
AX_PAUSE_TIME = 45
AX_MOVING = 46
AX_LOCK = 47
AX_PUNCH_L = 48
AX_PUNCH_H = 49

////////////////////////////////////// Status Return Levels

AX_RETURN_NONE = 0
AX_RETURN_READ = 1
AX_RETURN_ALL = 2

////////////////////////////////////// Instruction Set

AX_PING = 1
AX_READ_DATA = 2
AX_WRITE_DATA = 3
AX_REG_WRITE = 4
AX_ACTION = 5
AX_RESET = 6
AX_SYNC_WRITE = 131

////////////////////////////////////// Lengths

AX_RESET_LENGTH = 2
AX_ACTION_LENGTH = 2
AX_ID_LENGTH = 4
AX_LR_LENGTH = 4

```

AX_SRL_LENGTH = 4
AX_RDT_LENGTH = 4
AX_LEDALARM_LENGTH = 4
AX_SHUTDOWNALARM_LENGTH = 4
AX_TL_LENGTH = 4
AX_VL_LENGTH = 6
AX_AL_LENGTH = 7
AX_CM_LENGTH = 6
AX_CS_LENGTH = 5
AX_COMPLIANCE_LENGTH = 7
AX_CCW_CW_LENGTH = 8
AX_BD_LENGTH = 4
AX_TEM_LENGTH = 4
AX_MOVING_LENGTH = 4
AX_RWS_LENGTH = 4
AX_VOLT_LENGTH = 4
AX_LOAD_LENGTH = 4
AX_LED_LENGTH = 4
AX_TORQUE_LENGTH = 4
AX_POS_LENGTH = 4
AX_GOAL_LENGTH = 5
AX_MT_LENGTH = 5
AX_PUNCH_LENGTH = 5
AX_SPEED_LENGTH = 5
AX_GOAL_SP_LENGTH = 7

# //////////////////////////////////////// Specials
AX_BYTE_READ = 1
AX_INT_READ = 2
AX_ACTION_CHECKSUM = 250
AX_BROADCAST_ID = 254
AX_START = 255
AX_CCW_AL_L = 255
AX_CCW_AL_H = 3
AX_LOCK_VALUE = 1
LEFT = 0
RIGHT = 1
RX_TIME_OUT = 10
TX_DELAY_TIME = 0.000001 #0.00002

# RPi constants

```

```

RPI_DIRECTION_PIN = 18
RPI_DIRECTION_TX = GPIO.HIGH
RPI_DIRECTION_RX = GPIO.LOW
RPI_DIRECTION_SWITCH_DELAY = 0.000001 #0.000001

# static variables
port = None
gpioSet = False

def __init__(self):
    if(Ax12.port == None):
        Ax12.port = Serial("/dev/ttyAMA0", baudrate=1000000,
timeout=0.001)
    if(not Ax12.gpioSet):
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(Ax12.RPI_DIRECTION_PIN, GPIO.OUT)
        Ax12.gpioSet = True
        self.direction(Ax12.RPI_DIRECTION_RX)

connectedServos = []

# Error lookup dictionary for bit masking
dictErrors = { 1 : "Input Voltage",
                2 : "Angle Limit",
                4 : "Overheating",
                8 : "Range",
                16 : "Checksum",
                32 : "Overload",
                64 : "Instruction"
              }

# Custom error class to report AX servo errors
class axError(Exception) : pass

# Servo timeout
class timeoutError(Exception) : pass

def direction(self,d):
    GPIO.output(Ax12.RPI_DIRECTION_PIN, d)
    sleep(Ax12.RPI_DIRECTION_SWITCH_DELAY)

```

```

def readDataSimple(self):
    self.direction(Ax12.RPI_DIRECTION_RX)
    print Ax12.port.read()

def readData(self,id):
    self.direction(Ax12.RPI_DIRECTION_RX)
    sleep(0.00001) #0.0006
    reply = Ax12.port.read(5) # [0xff, 0xff, origin, length, error]
    try:
        assert ord(reply[0]) == 0xFF
    except:
        e = "Timeout on servo " + str(id)
        raise Ax12.timeoutError(e)

    try :
        length = ord(reply[3]) - 2
        error = ord(reply[4])

        if(error != 0):
            print "Error from servo: " + Ax12.dictErrors[error] + ' (code ' +
hex(error) + ')'
            return -error
        # just reading error bit
        elif(length == 0):
            return error
        else:
            if(length > 1):
                reply = Ax12.port.read(2)
                returnValue = (ord(reply[1])<<8) + (ord(reply[0])<<0)
            else:
                reply = Ax12.port.read(1)
                returnValue = ord(reply[0])
            return returnValue
    except Exception, detail:
        raise Ax12.axError(detail)

def ping(self,id):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()

```



```

checksum = (~(id + Ax12.AX_READ_DATA + Ax12.AX_PING))&0xff
outData = chr(Ax12.AX_START)
outData += chr(Ax12.AX_START)
outData += chr(id)
outData += chr(Ax12.AX_READ_DATA)
outData += chr(Ax12.AX_PING)
outData += chr(checksum)
Ax12.port.write(outData)
sleep(Ax12.TX_DELAY_TIME)
return self.readData(id)

def factoryReset(self, id, confirm = False):
    if(confirm):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        checksum = (~(id + Ax12.AX_RESET_LENGTH +
Ax12.AX_RESET))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_RESET_LENGTH)
        outData += chr(Ax12.AX_RESET)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)
    else:
        print "nothing done, please send confirm = True as this fuction
reset to the factory default value, i.e reset the motor ID"
        return

def setID(self, id, newId):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    checksum = (~(id + Ax12.AX_ID_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_ID + newId))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_ID_LENGTH)
    outData += chr(Ax12.AX_WRITE_DATA)

```

```

        outData += chr(Ax12.AX_ID)
        outData += chr(newId)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        #return self.readData(id)

    def setBaudRate(self, id, baudRate):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        br = ((2000000/long(baudRate))-1)
        checksum = (~((id + Ax12.AX_BD_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_BAUD_RATE + br))&0xff)
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_BD_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_BAUD_RATE)
        outData += chr(br)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)

    def setStatusReturnLevel(self, id, level):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        checksum = (~((id + Ax12.AX_SRL_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_RETURN_LEVEL + level))&0xff)
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_SRL_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_RETURN_LEVEL)
        outData += chr(level)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        #return self.readData(id)

```

```

def setReturnDelayTime(self, id, delay):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    checksum = (~(id + Ax12.AX_RDT_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_RETURN_DELAY_TIME +
(int(delay)/2)&0xff))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_RDT_LENGTH)
    outData += chr(Ax12.AX_WRITE_DATA)
    outData += chr(Ax12.AX_RETURN_DELAY_TIME)
    outData += chr((int(delay)/2)&0xff)
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    #return self.readData(id)

def lockRegister(self, id):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    checksum = (~(id + Ax12.AX_LR_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_LOCK +
Ax12.AX_LOCK_VALUE))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_LR_LENGTH)
    outData += chr(Ax12.AX_WRITE_DATA)
    outData += chr(Ax12.AX_LOCK)
    outData += chr(Ax12.AX_LOCK_VALUE)
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    #return self.readData(id)

def move(self, id, position):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    p = [position&0xff, position>>8]

```

```

        checksum = (~(id + Ax12.AX_GOAL_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_GOAL_POSITION_L + p[0] +
p[1]))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_GOAL_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_GOAL_POSITION_L)
        outData += chr(p[0])
        outData += chr(p[1])
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        #return self.readData(id)

def moveSpeed(self, id, position, speed):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    p = [position&0xff, position>>8]
    s = [speed&0xff, speed>>8]
    checksum = (~(id + Ax12.AX_GOAL_SP_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_GOAL_POSITION_L + p[0] + p[1] +
s[0] + s[1]))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_GOAL_SP_LENGTH)
    outData += chr(Ax12.AX_WRITE_DATA)
    outData += chr(Ax12.AX_GOAL_POSITION_L)
    outData += chr(p[0])
    outData += chr(p[1])
    outData += chr(s[0])
    outData += chr(s[1])
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    #return self.readData(id)

def moveRW(self, id, position):
    self.direction(Ax12.RPI_DIRECTION_TX)

```

```

    Ax12.port.flushInput()
    p = [position&0xff, position>>8]
    checksum = (~(id + Ax12.AX_GOAL_LENGTH +
Ax12.AX_REG_WRITE + Ax12.AX_GOAL_POSITION_L + p[0] +
p[1]))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_GOAL_LENGTH)
    outData += chr(Ax12.AX_REG_WRITE)
    outData += chr(Ax12.AX_GOAL_POSITION_L)
    outData += chr(p[0])
    outData += chr(p[1])
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    #return self.readData(id)

def moveSpeedRW(self, id, position, speed):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    p = [position&0xff, position>>8]
    s = [speed&0xff, speed>>8]
    checksum = (~(id + Ax12.AX_GOAL_SP_LENGTH +
Ax12.AX_REG_WRITE + Ax12.AX_GOAL_POSITION_L + p[0] + p[1] +
s[0] + s[1]))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_GOAL_SP_LENGTH)
    outData += chr(Ax12.AX_REG_WRITE)
    outData += chr(Ax12.AX_GOAL_POSITION_L)
    outData += chr(p[0])
    outData += chr(p[1])
    outData += chr(s[0])
    outData += chr(s[1])
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    #return self.readData(id)

```

```

def action(self):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(Ax12.AX_BROADCAST_ID)
    outData += chr(Ax12.AX_ACTION_LENGTH)
    outData += chr(Ax12.AX_ACTION)
    outData += chr(Ax12.AX_ACTION_CHECKSUM)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    # self.direction(Ax12.RPI_DIRECTION_RX)
    sleep(Ax12.TX_DELAY_TIME)

def setTorqueStatus(self, id, status):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    ts = 1 if ((status == True) or (status == 1)) else 0
    checksum = (~(id + Ax12.AX_TORQUE_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_TORQUE_STATUS + ts))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_TORQUE_LENGTH)
    outData += chr(Ax12.AX_WRITE_DATA)
    outData += chr(Ax12.AX_TORQUE_STATUS)
    outData += chr(ts)
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    #return self.readData(id)

def setLedStatus(self, id, status):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    ls = 1 if ((status == True) or (status == 1)) else 0
    checksum = (~(id + Ax12.AX_LED_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_LED_STATUS + ls))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)

```

```

        outData += chr(Ax12.AX_LED_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_LED_STATUS)
        outData += chr(Is)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        #return self.readData(id)

    def setTemperatureLimit(self, id, temp):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        checksum = (~(id + Ax12.AX_TL_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_LIMIT_TEMPERATURE +
temp))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_TL_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_LIMIT_TEMPERATURE)
        outData += chr(temp)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)

    def setVoltageLimit(self, id, lowVolt, highVolt):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        checksum = (~(id + Ax12.AX_VL_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_DOWN_LIMIT_VOLTAGE +
lowVolt + highVolt))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_VL_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_DOWN_LIMIT_VOLTAGE)
        outData += chr(lowVolt)
        outData += chr(highVolt)

```

```

        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)

    def setAngleLimit(self, id, cwLimit, ccwLimit):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        cw = [cwLimit&0xff, cwLimit>>8]
        ccw = [ccwLimit&0xff, ccwLimit>>8]
        checksum = (~(id + Ax12.AX_AL_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_CW_ANGLE_LIMIT_L + cw[0] +
cw[1] + ccw[0] + ccw[1]))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_AL_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_CW_ANGLE_LIMIT_L)
        outData += chr(cw[0])
        outData += chr(cw[1])
        outData += chr(ccw[0])
        outData += chr(ccw[1])
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        #return self.readData(id)

    def setTorqueLimit(self, id, torque):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        mt = [torque&0xff, torque>>8]
        checksum = (~(id + Ax12.AX_MT_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_MAX_TORQUE_L + mt[0] +
mt[1]))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_MT_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_MAX_TORQUE_L)

```



```

        outData += chr(mt[0])
        outData += chr(mt[1])
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)

def setPunchLimit(self, id, punch):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    p = [punch&0xff, punch>>8]
    checksum = (~(id + Ax12.AX_PUNCH_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_PUNCH_L + p[0] + p[1]))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_PUNCH_LENGTH)
    outData += chr(Ax12.AX_WRITE_DATA)
    outData += chr(Ax12.AX_PUNCH_L)
    outData += chr(p[0])
    outData += chr(p[1])
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    #return self.readData(id)

def setCompliance(self, id, cwMargin, ccwMargin, cwSlope,
ccwSlope):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    checksum = (~(id + Ax12.AX_COMPLIANCE_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_CW_COMPLIANCE_MARGIN +
cwMargin + ccwMargin + cwSlope + ccwSlope))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_COMPLIANCE_LENGTH)
    outData += chr(Ax12.AX_WRITE_DATA)
    outData += chr(Ax12.AX_CW_COMPLIANCE_MARGIN)
    outData += chr(cwMargin)
    outData += chr(ccwMargin)

```

```

        outData += chr(cwSlope)
        outData += chr(ccwSlope)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        #return self.readData(id)

    def setLedAlarm(self, id, alarm):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        checksum = (~(id + Ax12.AX_LEDALARM_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_ALARM_LED + alarm))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_LEDALARM_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_ALARM_LED)
        outData += chr(alarm)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)

    def setShutdownAlarm(self, id, alarm):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        checksum = (~(id + Ax12.AX_SHUTDOWNALARM_LENGTH +
Ax12.AX_WRITE_DATA + Ax12.AX_ALARM_SHUTDOWN +
alarm))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_SHUTDOWNALARM_LENGTH)
        outData += chr(Ax12.AX_WRITE_DATA)
        outData += chr(Ax12.AX_ALARM_SHUTDOWN)
        outData += chr(alarm)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)

```

```

def readTemperature(self, id):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    checksum = (~(id + Ax12.AX_TEM_LENGTH +
Ax12.AX_READ_DATA + Ax12.AX_PRESENT_TEMPERATURE +
Ax12.AX_BYTE_READ))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_TEM_LENGTH)
    outData += chr(Ax12.AX_READ_DATA)
    outData += chr(Ax12.AX_PRESENT_TEMPERATURE)
    outData += chr(Ax12.AX_BYTE_READ)
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    return self.readData(id)

def readPosition(self, id):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    checksum = (~(id + Ax12.AX_POS_LENGTH +
Ax12.AX_READ_DATA + Ax12.AX_PRESENT_POSITION_L +
Ax12.AX_INT_READ))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_POS_LENGTH)
    outData += chr(Ax12.AX_READ_DATA)
    outData += chr(Ax12.AX_PRESENT_POSITION_L)
    outData += chr(Ax12.AX_INT_READ)
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    return self.readData(id)

def readVoltage(self, id):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()

```

```

        checksum = (~(id + Ax12.AX_VOLT_LENGTH +
Ax12.AX_READ_DATA + Ax12.AX_PRESENT_VOLTAGE +
Ax12.AX_BYTE_READ))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_VOLT_LENGTH)
        outData += chr(Ax12.AX_READ_DATA)
        outData += chr(Ax12.AX_PRESENT_VOLTAGE)
        outData += chr(Ax12.AX_BYTE_READ)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)

def readSpeed(self, id):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    checksum = (~(id + Ax12.AX_SPEED_LENGTH +
Ax12.AX_READ_DATA + Ax12.AX_PRESENT_SPEED_L +
Ax12.AX_INT_READ))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)
    outData += chr(id)
    outData += chr(Ax12.AX_SPEED_LENGTH)
    outData += chr(Ax12.AX_READ_DATA)
    outData += chr(Ax12.AX_PRESENT_SPEED_L)
    outData += chr(Ax12.AX_INT_READ)
    outData += chr(checksum)
    Ax12.port.write(outData)
    sleep(Ax12.TX_DELAY_TIME)
    return self.readData(id)

def readLoad(self, id):
    self.direction(Ax12.RPI_DIRECTION_TX)
    Ax12.port.flushInput()
    checksum = (~(id + Ax12.AX_LOAD_LENGTH +
Ax12.AX_READ_DATA + Ax12.AX_PRESENT_LOAD_L +
Ax12.AX_INT_READ))&0xff
    outData = chr(Ax12.AX_START)
    outData += chr(Ax12.AX_START)

```

```

        outData += chr(id)
        outData += chr(Ax12.AX_LOAD_LENGTH)
        outData += chr(Ax12.AX_READ_DATA)
        outData += chr(Ax12.AX_PRESENT_LOAD_L)
        outData += chr(Ax12.AX_INT_READ)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)

    def readMovingStatus(self, id):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        checksum = (~(id + Ax12.AX_MOVING_LENGTH +
Ax12.AX_READ_DATA + Ax12.AX_MOVING
Ax12.AX_BYTE_READ)))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_MOVING_LENGTH)
        outData += chr(Ax12.AX_READ_DATA)
        outData += chr(Ax12.AX_MOVING)
        outData += chr(Ax12.AX_BYTE_READ)
        outData += chr(checksum)
        Ax12.port.write(outData)
        sleep(Ax12.TX_DELAY_TIME)
        return self.readData(id)

    def readRWStatus(self, id):
        self.direction(Ax12.RPI_DIRECTION_TX)
        Ax12.port.flushInput()
        checksum = (~(id + Ax12.AX_RWS_LENGTH +
Ax12.AX_READ_DATA + Ax12.AX_REGISTERED_INSTRUCTION +
Ax12.AX_BYTE_READ)))&0xff
        outData = chr(Ax12.AX_START)
        outData += chr(Ax12.AX_START)
        outData += chr(id)
        outData += chr(Ax12.AX_RWS_LENGTH)
        outData += chr(Ax12.AX_READ_DATA)
        outData += chr(Ax12.AX_REGISTERED_INSTRUCTION)
        outData += chr(Ax12.AX_BYTE_READ)

```

```
outData += chr(checksum)
Ax12.port.write(outData)
sleep(Ax12.TX_DELAY_TIME)
return self.readData(id)
```

```
def learnServos(self,minValue=1, maxValue=6, verbose=False) :
    servoList = []
    for i in range(minValue, maxValue + 1):
        try :
            temp = self.ping(i)
            servoList.append(i)
            if verbose: print "Found servo #" + str(i)
            time.sleep(0.1)

        except Exception, detail:
            if verbose : print "Error pinging servo #" + str(i) + ': ' + str(detail)
            pass
    return servoList
```

Anexo I Sensores ultrasonido raspberry pi

```
#!/usr/bin/python

import smbus
import time

#
=====
=====

# Distance_sensor_I2C Class

#=====
=====

# from Distance_sensor_I2C import Distance_sensor_I2C

class Distance_sensor(object):

    def __init__(self, bus_I2C = 1):
        self.address = 0x42
        self.bus = smbus.SMBus(bus_I2C)

    def read_byte(self, adr):
        time.sleep(0.0005)
        return self.bus.read_byte_data(self.address, adr)

    def read_word(self, adr):
        low = self.bus.read_byte_data(self.address, adr)
```

```
high = self.bus.read_byte_data(self.address, adr - 1)
val = (high << 8) + low
return val
```

```
def write_byte(self, adr, value):
    self.bus.write_byte_data(self.address, adr, value)
```

```
def read_distance(self):
    #number = self.bus.read_byte(self.address)
    flag = 1
    counter_t_out = 0
    while flag == 1 or counter_t_out > 40:
        sensor_data = []
        try:
            sensor_data.append(self.read_byte(0x01))
            sensor_data.append(self.read_byte(0x02))
            sensor_data.append(self.read_byte(0x03))
            sensor_data.append(self.read_byte(0x04))
            sensor_data.append(self.read_byte(0x05))
            flag = 0
        except:
            flag = 1
            counter_t_out = counter_t_out + 1
    return sensor_data
```


Anexo J Codigo procesamiento de imagenes.

```
# -*- coding: utf-8 -*-  
"""
```

```
Created on Thu Jul 21 14:50:33 2016
```

```
@author: eduardo  
"""
```

```
import cv2  
import numpy as np  
import math  
from lib2Motores import Ax12  
from serial import Serial  
from time import sleep
```

```
object_Ax12 = Ax12()  
object_Ax12.setAngleLimit(1,0x00,0x00)  
object_Ax12.setAngleLimit(2,0x00,0x00)
```

```
area_max = 150000  
area_min = 130000
```

```
w=420  
h=340
```

```
go = cv2.imread('Rgo1.jpg')  
stop = cv2.imread('Rstop1.jpg')  
giroder = cv2.imread('Rizq1.jpg')  
giroizq= cv2.imread('Rder.jpg')
```

```
imagen_referencia = [go,stop,giroizq,giroder]
```

```
def resize_y_binario(imagen):
```

```

img = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
#blur = cv2.GaussianBlur(img,(5,5),0)
#min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(blur)
#threshold = (min_val + max_val)/2

flag = True
try:
    ret, imagen_2 = cv2.threshold(img, 50, 255,
cv2.THRESH_BINARY)

    imagen_3 = cv2.resize(imagen_2,(w/2,h/2),interpolation =
cv2.INTER_AREA)

    #imagen_3 = cv2.resize(imagen_2,(w/2,h/2))
except:

    flag=False
if flag==True:
    return imagen_3
else:

    return False

def box_correction(box,center,angle):

    box2 = [[],[],[],[]]

    for i in range(0, 4):
        x = box[i][0] - center[0]
        y = box[i][1] - center[1]

        angle2 = (math.degrees(-math.atan2(y, x)) + angle) % 360
        r = math.sqrt(math.pow(x, 2) + math.pow(y, 2))

        posX = center[0] + r*math.cos(math.radians(angle2))
        posY = center[1] + r*math.sin(math.radians(-angle2))

        box2[i].append(int(posX))
        box2[i].append(int(posY))

```

```

box2 = np.array(box2)
return box2

```

```

def ssd(im_ref,crop):
    ssd_1 = []
    ssd = 0
    for i in range(0,4):

        a = im_ref[i]
        o2 = resize_y_binario(a)

        ssd = 0
        for M in range(h/2):
            for N in range(w/2):

                ssd += abs(int(crop[M,N])-int(o2[M,N]))/100
        ssd_1.append(ssd)
    ssd_1 = np.array(ssd_1)
    return ssd_1

```

```

def getthresholdedimg(hsv):
    threshImg =
    cv2.inRange(hsv,np.array((cv2.getTrackbarPos('Hue_Low','Trackbars'),
    cv2.getTrackbarPos('Saturation_Low','Trackbars'),cv2.getTrackbarPos('
    Value_Low','Trackbars'))),np.array((cv2.getTrackbarPos('Hue_High','Tra
    ckbars'),cv2.getTrackbarPos('Saturation_High','Trackbars'),cv2.getTrack
    barPos('Value_High','Trackbars'))))
    return threshImg

```

```

def getTrackValue(value):
    return value

```

```

cv2.namedWindow('Output')
cv2.namedWindow('Trackbars', cv2.WINDOW_NORMAL)
cv2.createTrackbar('Hue_Low','Trackbars',0,255, getTrackValue)
cv2.createTrackbar('Saturation_Low','Trackbars',0,255, getTrackValue)

```

```

cv2.createTrackbar('Value_Low','Trackbars',0,255, getTrackValue)

cv2.createTrackbar('Hue_High','Trackbars',0,255, getTrackValue)
cv2.createTrackbar('Saturation_High','Trackbars',0,255, getTrackValue)
cv2.createTrackbar('Value_High','Trackbars',0,255, getTrackValue)
cv2.createTrackbar('Caliberate','Trackbars',0,1, getTrackValue)


cv2.setTrackbarPos('Hue_Low', 'Trackbars', 19)
cv2.setTrackbarPos('Saturation_Low', 'Trackbars', 135)
cv2.setTrackbarPos('Value_Low', 'Trackbars', 66)
cv2.setTrackbarPos('Hue_High', 'Trackbars', 56)
cv2.setTrackbarPos('Saturation_High', 'Trackbars', 255)
cv2.setTrackbarPos('Value_High', 'Trackbars', 255)


video = cv2.VideoCapture(0)
width,height = video.get(3),video.get(4)
print "PROGRAMA INICIANDO ...."


while(1):
    __,imagen_original = video.read()
    imagen_original = cv2.flip(imagen_original,1)
    hsv = cv2.cvtColor(imagen_original,cv2.COLOR_BGR2HSV)
    thrImg = getthresholdedimg(hsv)
    erode = cv2.erode(thrImg,None,iterations = 5)
    dilate = cv2.dilate(erode,None,iterations = 5)


    center = (w/2,h/2)


    __, contours, __ =
cv2.findContours(dilate,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE
)

```

```

for cnt in contours:
    area = cv2.contourArea(cnt)

    print"areaaaaaaaaaaaaaaaaa",area

    if area < area_max and area > area_min:
        print "area",area
        print "-->"
        im = imagen_original.copy()
        rect = cv2.minAreaRect(cnt)
        box = cv2.boxPoints(rect)
        box = np.int0(box)

        giro = cv2.fitEllipse(cnt)

        angle = (giro[2] - 90) % 360

        M = cv2.getRotationMatrix2D(center, angle, 1)
        dst = cv2.warpAffine(im,M,(w,h))

        box2 = box_correction(box,center,angle)

        x1,y1,w1,h1 = cv2.boundingRect(box2)
        cx,cy = x1+w1/2, y1+h1/2

        crop_img = dst[y1: y1 + h1, x1: x1 + w1]

        crop_ct = resize_y_binario(crop_img)

        if crop_ct is not False:
            go,stop,giroI,giroD = ssd(imagen_referencia,crop_ct)

            print "go ",go
            print "Stop ",stop
            print "giroIII ",giroI

```

```

        print "giroDDD ",giroD
#        if giroD < 6500:
#            print "          giroDDD"
#            if girol < 5500:
#                print "          girolII"
#                if go < 6000:
#                    print "          go"
#                    if stop < 10000:
#                        print "          stop"

        object_Ax12.moveSpeed(1,0x00,0x00)
        object_Ax12.moveSpeed(2,0x00,0x130)

        sleep(5.19)
    else:
        object_Ax12.moveSpeed(1,0x00,0x496)
        object_Ax12.moveSpeed(2,0x00,0x9a)

        time.sleep(0.5)
else:
    object_Ax12.moveSpeed(1,0x00,0x496)
    object_Ax12.moveSpeed(2,0x00,0x9a)

if(cv2.getTrackbarPos('Caliberate','Trackbars') == 1):
    cv2.imshow('Output',thrImg)
else:
    cv2.imshow('Output',imagen_original)
#qcv2.waitKey(50)
if cv2.waitKey(10) & 0xFF == ord('k'):
    break

cv2.destroyAllWindows()
video.release()

```

Anexo K Algoritmo evasión de obstáculos

```
from threading import Timer, Thread, Event
from calDist import calDist

from time import sleep

from lib2Motores import Ax12
from serial import Serial
from MPU3 import mpu6050
from Distance_sensor import Distance_sensor
from HMC5983 import HMC5983

import cv2

import math
import numpy as np
from sympy import *
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

def guardar(nombre_archivo, coor):

    archivo = open(nombre_archivo, "a")
    for posX, posY in coor:
        archivo.write(str(posX)+", "+str(posY)+"\n")
    archivo.close()

def processLine(line):
    sen_compass = HMC5983()

    # print vel
    x = Symbol('x')
    array[4] = 1
    i = 0
    d = -0.1
    x_2 = 0
    y_2 = 0
    while not array[2].wait(0.1):
```

```
sen_bruj = sen_compass.get_angle()
```

```
wi = array[0]  
wd = array[1]
```

```
p = (2.6/2)*(wi+wd)  
theta_1 = (2.6/(2*17))*(wd-wi)  
theta_2 = integrate(theta_1, (x, 0, i))  
x_2 += integrate(math.sin(round(sen_bruj[0],2))*p, (x,d,i))  
y_2 += integrate(math.cos(round(sen_bruj[0],2))*p, (x,d,i))  
x3 = round(x_2,1)  
y3 = round(y_2,1)  
coordenadas = [(x3,y3)]  
dat = guardar('carrito.txt',coordenadas)  
i=i+0.1  
d =d+0.1
```

```
return
```

```
cv2.namedWindow('frame_pru', cv2.WINDOW_AUTOSIZE)  
wi = [1.98 , 1.40 , 0.87 , 0.34,0,-1.98]  
wd = [1.98 , 1.40 , 0.87 , 0.34,0,-1.98]  
stopFlag = Event()  
distance = 0  
cont = 0  
bruj = 0  
array = [wi[0],wd[0] ,stopFlag, distance,cont,bruj]
```

```
processThread = Thread(target=processLine, args=(array,))  
processThread.start()
```

```
object_Ax12 = Ax12()
```

```
object_Ax12.setAngleLimit(1,0x00,0x00)  
object_Ax12.setAngleLimit(2,0x00,0x00)
```



```

sen_dis = Distance_sensor()
sen_mpu = mpu6050(0x68)
sen_compass = HMC5983()

```

```

while True :

```

```

    ultras = sen_dis.read_distance()
    accel_data = sen_mpu.get_accel_data()
    gyro_data = sen_mpu.get_gyro_data()

```

```

    if ultras[0]==0:
        ultras[0]=128
    if ultras[1]==0:
        ultras[1]=128
    if ultras[2]==0:
        ultras[2]=128
    if ultras[3]==0:
        ultras[3]=128
    if ultras[4]==0:
        ultras[4]=128

```

```

    if accel_data['y']>2 or accel_data['x']>2 or accel_data['y']<-2 or
    accel_data['x']<-2 :
        array[0]=wi[4]
        array[1]=wd[4]

```

```

        object_Ax12.moveSpeed(1,0x00,0x00)
        object_Ax12.moveSpeed(2,0x00,0x00)
    else:

```

```

        if ultras[0]<10 and ultras[1]<20 and ultras[2]<17 and
        ultras[3]<20 and ultras[4]<10:
            print "caso: 1"
            array[0]=wi[5]
            array[1]=wd[5]

```

```

        object_Ax12.moveSpeed(1,0x00,0x130)
        object_Ax12.moveSpeed(2,0x00,0x52c)
    sleep(5)

    elif ultras[1]<20 and ultras[2]<17 and ultras[3]<20 and
ultras[4]<10:
    print "caso : 2"
        array[0]=wi[0]
        array[1]=wd[4]

        object_Ax12.moveSpeed(1,0x00,0x00)
        object_Ax12.moveSpeed(2,0x00,0x130)

    elif ultras[0]<10 and ultras[1]<20 and ultras[2]<17 and
ultras[3]<20:
    print"caso: 3"
        array[0]=wi[4]
        array[1]=wd[0]

        object_Ax12.moveSpeed(1,0x00,0x52c)
        object_Ax12.moveSpeed(2,0x00,0x00)

    elif ultras[0]<10 and ultras[1]<20 and ultras[3]<20 and
ultras[4]<10:
    print"caso: 4"
        array[0]=wi[0]
        array[1]=wd[0]

        object_Ax12.moveSpeed(1,0x00,0x52c)
        object_Ax12.moveSpeed(2,0x00,0x130)

    elif ultras[0]<10 and ultras[1]<20 and ultras[2]<17:
        print"caso: 5"
        array[0]=wi[3]
        array[1]=wd[0]

        object_Ax12.moveSpeed(1,0x00,0x52c)
        object_Ax12.moveSpeed(2,0x00,0x4f)

    elif ultras[2]<17 and ultras[3]<20 and ultras[4]<10:
        print"caso: 6"

```

```

array[0]=wi[0]
array[1]=wd[3]

        object_Ax12.moveSpeed(1,0x00,0x44b)
        object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[1]<21 and ultras[2]<18 and ultras[3]<21:
    print"caso: 10"

if ultras[0]<5 and ultras[4]>20:
    array[0]=wi[4]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x00)

else:
    array[0]=wi[0]
    array[1]=wd[4]

    object_Ax12.moveSpeed(1,0x00,0x00)
    object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[4]<11 and ultras[3]<20:
    print"caso: 7"
    array[0]=wi[0]
    array[1]=wd[2]

    object_Ax12.moveSpeed(1,0x00,0x496)
    object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[0]<11 and ultras[1]<20:
    print"caso: 8"
    array[0]=wi[2]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)

```

```

        object_Ax12.moveSpeed(2,0x00,0x9a)

    elif ultras[0]<8 and ultras[4]<8:
        print"caso: 9"
        array[0]=wi[0]
        array[1]=wd[0]

        object_Ax12.moveSpeed(1,0x00,0x52c)
        object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[1]<20 and ultras[2]<17 :
    print "caso:11"
    array[0]=wi[3]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x4f)

elif ultras[2]<17 and ultras[3]<20 :
    print "caso: 12"
    array[0]=wi[0]
    array[1]=wd[3]

    object_Ax12.moveSpeed(1,0x00,0x44b)
    object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[1]<20:
    print"sensor1"
    array[0]=wi[2]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x9a)
elif ultras[3]<20:
    print"sensor3"
    array[0]=wi[0]
    array[1]=wd[2]

    object_Ax12.moveSpeed(1,0x00,0x496)
    object_Ax12.moveSpeed(2,0x00,0x130)

```

```

elif ultras[2]<17:
print"caso: 13"

    if ultras[0]<ultras[4]:
        array[0]=wi[4]
        array[1]=wd[0]

        object_Ax12.moveSpeed(1,0x00,0x52c)
        object_Ax12.moveSpeed(2,0x00,0x00)

    else:
        array[0]=wi[0]
        array[1]=wd[4]

        object_Ax12.moveSpeed(1,0x00,0x00)
        object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[0]<9:
    array[0]=wi[1]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x103)
print"sensor 0"
elif ultras[4]<9:
    array[0]=wi[1]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x4ff)
    object_Ax12.moveSpeed(2,0x00,0x130)
print"sensor 4"

else:
    array[0]=wi[0]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)

```

```
        object_Ax12.moveSpeed(2,0x00,0x130)
    print"no hay ningun caso"
    print ultras
sleep(0.1)
```

Anexo L CÓDIGO COMPLETO

```
import cv2
import numpy as np
import math
from lib2Motores import Ax12
from serial import Serial
from time import sleep

from threading import Timer, Thread, Event
from calDist import calDist

from MPU3 import mpu6050
from Distance_sensor import Distance_sensor
from HMC5983 import HMC5983

from sympy import *
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

area_max = 150000
area_min = 130000

w=420
h=340

go = cv2.imread('Rgo1.jpg')
stop = cv2.imread('Rstop1.jpg')
giroder = cv2.imread('Rizq1.jpg')
giroizq = cv2.imread('Rder.jpg')

imagen_referencia = [go, stop, giroizq, giroder]

def resize_y_binario(imagen):

    img = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    #blur = cv2.GaussianBlur(img, (5,5), 0)
```

```

#min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(blur)
#threshold = (min_val + max_val)/2

flag = True
try:
    ret, imagen_2 = cv2.threshold(img, 50, 255, cv2.THRESH_BINARY)

    imagen_3 = cv2.resize(imagen_2,(w/2,h/2),interpolation = cv2.INTER_AREA)

    #imagen_3 = cv2.resize(imagen_2,(w/2,h/2))
except:

    flag=False
    if flag==True:
        return imagen_3
    else:

        return False

def box_correction(box,center,angle):

    box2 = [[],[],[],[ ]]

    for i in range(0, 4):
        x = box[i][0] - center[0]
        y = box[i][1] - center[1]

        angle2 = (math.degrees(-math.atan2(y, x)) + angle) % 360
        r = math.sqrt(math.pow(x, 2) + math.pow(y, 2))

        posX = center[0] + r*math.cos(math.radians(angle2))
        posY = center[1] + r*math.sin(math.radians(-angle2))

        box2[i].append(int(posX))
        box2[i].append(int(posY))

    box2 = np.array(box2)
    return box2

def ssd(im_ref,crop):

```



```

ssd_1 = []
ssd = 0
for i in range(0,4):

    a = im_ref[i]
    o2 = resize_y_binario(a)

    ssd = 0
    for M in range(h/2):
        for N in range(w/2):

            ssd += abs(int(crop[M,N])-int(o2[M,N]))/100
    ssd_1.append(ssd)
ssd_1 = np.array(ssd_1)
return ssd_1

def getthresholdedimg(hsv):
    threshImg =
cv2.inRange(hsv,np.array((cv2.getTrackbarPos('Hue_Low','Trackbars'),cv2.getTrackbarPos('Saturation_Low','Trackbars'),cv2.getTrackbarPos('Value_Low','Trackbars'))),np.array((cv2.getTrackbarPos('Hue_High','Trackbars'),cv2.getTrackbarPos('Saturation_High','Trackbars'),cv2.getTrackbarPos('Value_High','Trackbars'))))
    return threshImg

def getTrackValue(value):
    return value

def guardar(nombre_archivo,coor):

    archivo = open(nombre_archivo, "a")
    for posX,posY in coor:
        archivo.write(str(posX)+","+str(posY)+"\n")
    archivo.close()

def processLine(line):
    sen_compass = HMC5983()

    # print vel
    x = Symbol('x')

```

```

array[4] = 1
i = 0
d = -0.1
x_2 = 0
y_2 = 0
while not array[2].wait(0.1):
    sen_bruj = sen_compass.get_angle()

    wi = array[0]
    wd = array[1]

    p = (2.6/2)*(wi+wd)
    theta_1 = (2.6/(2*17))*(wd-wi)
    theta_2 = integrate(theta_1, (x, 0, i))
    x_2 += integrate(math.sin(round(sen_bruj[0],2))*p, (x,d,i))
    y_2 += integrate(math.cos(round(sen_bruj[0],2))*p, (x,d,i))
    x3 = round(x_2,1)
    y3 = round(y_2,1)
    coordenadas = [(x3,y3)]
    dat = guardar('carrito.txt',coordenadas)
    i=i+0.1
    d=d+0.1

return

```

```

cv2.namedWindow('Output')
cv2.namedWindow('Trackbars', cv2.WINDOW_NORMAL)
cv2.createTrackbar('Hue_Low', 'Trackbars', 0, 255, getTrackValue)
cv2.createTrackbar('Saturation_Low', 'Trackbars', 0, 255, getTrackValue)
cv2.createTrackbar('Value_Low', 'Trackbars', 0, 255, getTrackValue)

cv2.createTrackbar('Hue_High', 'Trackbars', 0, 255, getTrackValue)
cv2.createTrackbar('Saturation_High', 'Trackbars', 0, 255, getTrackValue)
cv2.createTrackbar('Value_High', 'Trackbars', 0, 255, getTrackValue)
cv2.createTrackbar('Caliberate', 'Trackbars', 0, 1, getTrackValue)

```

```

cv2.setTrackbarPos('Hue_Low', 'Trackbars', 19)
cv2.setTrackbarPos('Saturation_Low', 'Trackbars', 135)
cv2.setTrackbarPos('Value_Low', 'Trackbars', 66)
cv2.setTrackbarPos('Hue_High', 'Trackbars', 56)
cv2.setTrackbarPos('Saturation_High', 'Trackbars', 255)
cv2.setTrackbarPos('Value_High', 'Trackbars', 255)

video = cv2.VideoCapture(0)
width,height = video.get(3),video.get(4)
print "PROGRAMA INICIANDO ...."

wi = [1.98 , 1.40 , 0.87 , 0.34,0,-1.98]
wd = [1.98 , 1.40 , 0.87 , 0.34,0,-1.98]
stopFlag = Event()
distance = 0
cont = 0
bruj = 0
array = [wi[0],wd[0] ,stopFlag, distance,cont,bruj]

processThread = Thread(target=processLine, args=(array,))
processThread.start()


object_Ax12 = Ax12()
object_Ax12.setAngleLimit(1,0x00,0x00)
object_Ax12.setAngleLimit(2,0x00,0x00)
sen_dis = Distance_sensor()
sen_mpu = mpu6050(0x68)
sen_compass = HMC5983()

while(1):
    _imagen_original = video.read()
    imagen_original = cv2.flip(imagen_original,1)
    hsv = cv2.cvtColor(imagen_original,cv2.COLOR_BGR2HSV)
    thrImg = getthresholdedimg(hsv)
    erode = cv2.erode(thrImg,None,iterations = 5)
    dilate = cv2.dilate(erode,None,iterations = 5)

```

```

ultras = sen_dis.read_distance()
accel_data = sen_mpu.get_accel_data()
gyro_data = sen_mpu.get_gyro_data()

if ultras[0]==0:
    ultras[0]=128
if ultras[1]==0:
    ultras[1]=128
if ultras[2]==0:
    ultras[2]=128
if ultras[3]==0:
    ultras[3]=128
if ultras[4]==0:
    ultras[4]=128

center = (w/2,h/2)

_, contours, _ =
cv2.findContours(dilate,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    area = cv2.contourArea(cnt)

    print"areaaaaaaaaaaaaaaa",area

    if area < area_max and area > area_min:
        print "area",area
        print "-->"
        im = imagen_original.copy()
        rect = cv2.minAreaRect(cnt)
        box = cv2.boxPoints(rect)
        box = np.int0(box)

        giro = cv2.fitEllipse(cnt)

```

```

angle = (giro[2] - 90) % 360

M = cv2.getRotationMatrix2D(center, angle, 1)
dst = cv2.warpAffine(im,M,(w,h))

box2 = box_correction(box,center,angle)

x1,y1,w1,h1 = cv2.boundingRect(box2)
cx,cy = x1+w1/2, y1+h1/2

crop_img = dst[y1: y1 + h1, x1: x1 + w1]

crop_ct = resize_y_binario(crop_img)

if crop_ct is not False:
    go,stop,giroI,giroD = ssd(imagen_referencia,crop_ct)

    print "go ",go
    print "Stop ",stop
    print "giroIII ",giroI
    print "giroDDD ",giroD
    if giroD < 9000:
        print "          giroDDD"
        array[0]=wi[0]
        array[1]=wd[4]
        object_Ax12.moveSpeed(1,0x00,0x00)
        object_Ax12.moveSpeed(2,0x00,0x130)
        sleep(5.24)

        if giroI < 10000:
            print "          giroIII"
            array[0]=wi[4]
            array[1]=wd[0]
            object_Ax12.moveSpeed(1,0x00,0x52c)
            object_Ax12.moveSpeed(2,0x00,0x00)
            sleep(5.24)
            if go < 10000:
                print "          go"

```

```

array[0]=wi[0]
array[1]=wd[5]
object_Ax12.moveSpeed(1,0x00,0x130)
object_Ax12.moveSpeed(2,0x00,0x130)
sleep(5.2)

sleep(100)
if stop < 10000:
    print "          stop"
array[0]=wi[4]
array[1]=wd[4]
object_Ax12.moveSpeed(1,0x00,0x00)
object_Ax12.moveSpeed(2,0x00,0x00)

sleep(100)

if accel_data['y']>2 or accel_data['x']>2 or accel_data['y']<-2 or accel_data['x']<-2 :
    array[0]=wi[4]
    array[1]=wd[4]

    object_Ax12.moveSpeed(1,0x00,0x00)
    object_Ax12.moveSpeed(2,0x00,0x00)
else:

    if ultras[0]<10 and ultras[1]<20 and ultras[2]<17 and ultras[3]<20 and
ultras[4]<10:
        print "caso: 1"
        array[0]=wi[5]
        array[1]=wd[5]

        object_Ax12.moveSpeed(1,0x00,0x130)
        object_Ax12.moveSpeed(2,0x00,0x52c)
        sleep(5)

        elif ultras[1]<20 and ultras[2]<17 and ultras[3]<20 and ultras[4]<10:
            print "caso : 2"
            array[0]=wi[0]
            array[1]=wd[4]

```

```

        object_Ax12.moveSpeed(1,0x00,0x00)
    object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[0]<10 and ultras[1]<20 and ultras[2]<17 and ultras[3]<20:
    print"caso: 3"
    array[0]=wi[4]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x00)

elif ultras[0]<10 and ultras[1]<20 and ultras[3]<20 and ultras[4]<10:
    print"caso: 4"
    array[0]=wi[0]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[0]<10 and ultras[1]<20 and ultras[2]<17:
    print"caso: 5"
    array[0]=wi[3]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x4f)

elif ultras[2]<17 and ultras[3]<20 and ultras[4]<10:
    print"caso: 6"
    array[0]=wi[0]
    array[1]=wd[3]

    object_Ax12.moveSpeed(1,0x00,0x44b)
    object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[1]<21 and ultras[2]<18 and ultras[3]<21:
    print"caso: 10"

    if ultras[0]<5 and ultras[4]>20:
        array[0]=wi[4]

```

```

        array[1]=wd[0]

        object_Ax12.moveSpeed(1,0x00,0x52c)
        object_Ax12.moveSpeed(2,0x00,0x00)

    else:
        array[0]=wi[0]
        array[1]=wd[4]

        object_Ax12.moveSpeed(1,0x00,0x00)
        object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[4]<11 and ultras[3]<20:
    print"caso: 7"
    array[0]=wi[0]
    array[1]=wd[2]

    object_Ax12.moveSpeed(1,0x00,0x496)
    object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[0]<11 and ultras[1]<20:
    print"caso: 8"
    array[0]=wi[2]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x9a)

    elif ultras[0]<8 and ultras[4]<8:
        print"caso: 9"
        array[0]=wi[0]
        array[1]=wd[0]

        object_Ax12.moveSpeed(1,0x00,0x52c)
        object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[1]<20 and ultras[2]<17 :
    print "caso:11"
    array[0]=wi[3]

```



```

array[1]=wd[0]

object_Ax12.moveSpeed(1,0x00,0x52c)
object_Ax12.moveSpeed(2,0x00,0x4f)

elif ultras[2]<17 and ultras[3]<20 :
    print "caso: 12"
    array[0]=wi[0]
    array[1]=wd[3]

    object_Ax12.moveSpeed(1,0x00,0x44b)
    object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[1]<20:
    print"sensor1"
    array[0]=wi[2]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x9a)
elif ultras[3]<20:
    print"sensor3"
    array[0]=wi[0]
    array[1]=wd[2]

    object_Ax12.moveSpeed(1,0x00,0x496)
    object_Ax12.moveSpeed(2,0x00,0x130)

    elif ultras[2]<17:
        print"caso: 13"

if ultras[0]<ultras[4]:
    array[0]=wi[4]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x00)

else:
    array[0]=wi[0]
    array[1]=wd[4]

```

```

        object_Ax12.moveSpeed(1,0x00,0x00)
        object_Ax12.moveSpeed(2,0x00,0x130)

elif ultras[0]<9:
    array[0]=wi[1]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x103)
    print"sensor 0"
    elif ultras[4]<9:
array[0]=wi[1]
array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x4ff)
    object_Ax12.moveSpeed(2,0x00,0x130)
    print"sensor 4"

else:
    array[0]=wi[0]
    array[1]=wd[0]

    object_Ax12.moveSpeed(1,0x00,0x52c)
    object_Ax12.moveSpeed(2,0x00,0x130)

    print"no hay ningun caso"
    print ultras
sleep(0.1)

if(cv2.getTrackbarPos('Caliberate','Trackbars') == 1):
    cv2.imshow('Output',thrImg)
else:
    cv2.imshow('Output',imagen_original)
#qcv2.waitKey(50)
if cv2.waitKey(10) & 0xFF == ord('k'):
    break

```

```
cv2.destroyAllWindows()  
video.release()
```